KU LEUVEN

ARENBERG DOCTORAL SCHOOL
FACULTY OF ENGINEERING SCIENCE

Relational Affordances and their Applications

Bogdan Moldovan

Supervisor:
Prof. dr. Luc De Raedt

Dissertation presented in partial fulfillment of the requirements for the degree of PhD in Engineering

March 2015

RELATIONAL AFFORDANCES AND THEIR APPLICATIONS

Bogdan MOLDOVAN

Supervisor:
Prof. dr. Luc De Raedt

Members of the
Examination committee:
Prof. dr. ir. Paul Van Houtte, chair
Prof. dr. Luc De Raedt, supervisor
Prof. dr. ir. Herman Bruyninckx
Prof. dr. ir. Tinne Tuytelaars
Prof. dr. Claude Sammut
(University of New South Wales, Australia)
Prof. dr. José Santos-Victor
(Instituto Superior Técnico, Portugal)
Prof. dr. Martijn van Otterlo
(Nijmegen, The Netherlands)

Dissertation presented in
partial fulfillment of the
requirements for the
degree of PhD in Engineering

March 2015

Abstract

The concept of affordances is used in robotics to model action opportunities of a robot on objects in the environment, and as such they play a role in building basic cognitive capabilities of the robot. Affordances generally model isolated objects in the environment and capture the interdependencies between object properties, executed actions on those objects, and the effects of those respective actions. However, many real-world scenarios involve configurations of multiple objects that interact with each other when manipulated.

This thesis proposes the use of recent advances in statistical relational learning to build relational affordance models, where the (spatial) relations between the different objects, such as relative distances between pairs of objects, are taken into account. Two-object interaction models are learned from the robot interacting with the objects in the world in a behavioural babbling stage. These models can then be employed in situations with arbitrary numbers of objects. This model thus generalizes over objects and can deal effectively with uncertainty.

To show the relevance of relational affordance models, we further investigate the use of relational affordances in three different applications. These applications are illustrated by the use of the iCub and PR2 robots in simulation, and the use of the iCub robot in a real setting.

Firstly, we use SRL methods to create a relational affordance model for two-arm robot actions, for settings where these can be approximated by a combination of the two single-arm actions composing them. The arms may act simultaneously or sequentially, and the robot is given background knowledge about possible actions in its environment. SRL is used here to generalise and build a higher-level model for a set of two-arm actions settings in a household environment, by using symmetries of the arms to model effects of actions executed with the arm not involved in the behavioural babbling stage, and background knowledge about two-arm actions.

Secondly, we employ relational affordance models in a planning setting with high-level goals which are specified by (spatial) relations between the objects. The relational affordance model can be used to define a state transition model in a table-top setting, and we provide a planning algorithm to be used to infer the next best possible action for the robot to execute towards reaching the given goal. The robot is then given a series of planning tasks of increasing difficulty in a table-top environment, which it needs to solve.

Finally, to tackle an occluded object search task, we use the concept of relational affordances to search for any object affording a given action in a kitchen environment with many shelves. Multiple object types can afford the action, and each type allows for many different objects with size and shape modelled by probability distributions, thus relaxing some of the previous assumptions in the field. Moreover, we allow for stacked objects, a more realistic modelling of objects in shelves, introducing more complex object spatial relations.

Beknopte samenvatting

In de robotica worden affordances gebruikt voor het modelleren van mogelijke acties die een robot kan uitvoeren met een bepaald object in de omgeving. Ze spelen daarom een belangrijke rol in de opbouw van de primaire cognitieve vermogens van de robot. Over het algemeen modelleren affordances geïsoleerde objecten in de omgeving en beschrijven ze de relatie tussen de eigenschappen van een object, acties die op het object uitgevoerd kunnen worden, en de effecten van deze acties. In realiteit treden echter vaak scenario's op waarbij meerdere objecten met elkaar interageren.

Deze thesis stelt het gebruik voor van recente ontwikkelingen in statistisch relationeel leren (SRL) voor het opbouwen van relationele affordance modellen, waarbij rekening wordt gehouden met de (ruimtelijke) relaties tussen de verschillende objecten, zoals de relatieve afstanden tussen objectparen. Interactiemodellen voor twee objecten worden geleerd uit interactie van de robot met de objecten in de wereld tijdens een gedragsbrabbelfase. Deze modellen kunnen dan worden toegepast in situaties met een willekeurig aantal objecten. Dit model generaliseert dus over objecten en kan effectief omgaan met onzekerheden.

Om de relevantie van relationele affordance modellen aan te tonen, onderzochten we verder het gebruik van relationele affordances in drie verschillende toepassingen. Deze toepassingen worden geïllustreerd door het gebruik van de iCub en PR2 robots in simulatie, en het gebruik van de iCub robot in een echte setting.

Ten eerste, gebruiken we SRL methoden om een relationeel affordance model voor twee-armige robotacties te maken, in situaties waar deze kunnen worden benaderd door een combinatie van twee enkelarmige acties. De armen kunnen gelijktijdig gebruikt worden of in sequentie, en de robot heeft toegang tot achtergrondkennis over mogelijke acties in zijn omgeving. SRL wordt hier gebruikt om te generaliseren en om een hoger niveau model op te bouwen

voor twee-armige acties in een huiselijke omgeving, waarbij gebruik gemaakt wordt van symmetrieën tussen de armen om effecten te modelleren van acties uitgevoerd met de arm die niet betrokken is in de gedragsbrabbelfase, en achtergrondkennis over twee-armige acties.

Ten tweede, maken we gebruik van relationele affordance modellen in een planningsomgeving met hoog niveau doelen die worden gespecificeerd door (ruimtelijke) relaties tussen de objecten. Het relationele affordance model kan gebruikt worden om een toestandsovergangsmodel in een tafelblad omgeving te definiëren. We bieden een planningsalgoritme voor het bepalen van de best mogelijke volgende actie uit te voeren door de robot met het oog op het bereiken van het gegeven doel. De robot wordt dan een reeks van planningstaken gegeven van toenemende moeilijkheidsgraad in een tafelblad omgeving die opgelost moeten worden.

Tenslotte, gebruiken we relationele affordances om te zoeken naar een object dat een gegeven actie aanbiedt in een keuken omgeving met veel planken. Meerdere types objecten kunnen de actie bieden, en elke type object zorgt voor veel verschillende voorwerpen met afmetingen en vorm gemodelleerd door kansverdelingen. Hierbij relaxeren we sommige van de eerdere veronderstellingen in het vakgebied. Bovendien laten we voor gestapelde voorwerpen een meer realistische modellering van objecten in schappen toe, waarbij we meer complexe object ruimtelijke relaties introduceren.

Acknowledgments

Pursuing this Ph.D. has been a challenging, exciting and rewarding experience. Many people were involved along the way, and through their ideas, support and advice helped me achieve this goal. I would like to thank all of them here for their contribution.

First of all, I would like to thank my supervisor Luc De Raedt. He encouraged me to look for ideas, ask the right questions, and inspired me with pointers and suggestions. We explored together research ideas in the field of robotics, and I am thankful for his support and encouragement throughout all the stages of my Ph.D. at KU Leuven.

Secondly, I want to kindly thank all the members of my jury, Herman Bruyninckx, Tinne Tuytelaars, Claude Sammut, José Santos Victor, and Martijn van Otterlo for carefully reading this text, and for the constructive feedback which helped me improve it. Special thanks go to Martijn van Otterlo who as a postdoc at KU Leuven also guided my first research steps in this field. I would also like to thank Prof. Paul Van Houtte for chairing my defence.

Over the years, I collaborated with many people in order to publish all the papers on which this thesis is based, and so I would want to thank here all of my co-authors throughout my Ph.D. studies: Davide, Ingo, Jesse, José, Laura, Luc, Martijn, McElory, and Plinio. Your input was invaluable, and had an impact on the final shape of all our papers. They would not have been the same without you.

Of course, pursuing a Ph.D. would not have been the same without a stimulating, and enjoyable work environment, and for this I am thankful to all the past and present members of the DTAI research group. I would like to thank Jesse Davis for his input in the early stages of my Ph.D. studies. Next, I would want to thank all the people with whom I had the pleasure of sharing an office during my time at KU Leuven: Aaron, Davide, Laura, Martijn, Mathias, McElory, and Parisa, for our many interesting discussions and the exchange of research ideas,

and for the tea-breaks during the day. I am especially thankful to Davide for showing me how to use his Distributional Clauses, which I used in several of my experiments, and for his Italian cookies. During my time at KU Leuven, I also enjoyed the daily lunches with all the current and former members of our "Alma-group". Our regular lunches were a bright spot every day throughout all these years. And during these years I also enjoyed the various activities around Leuven and Belgium with several past and present members of DTAI: Andrew, Antoine, Anton, Benjamin, Clement, Daniele, Davide, Gitte, Irma, Jessa, Kurt, McElory, Sebastijan, Sergey, Siegfried, Vladimir, and others. By the same token, I am grateful for all the friends from Europe and all over the world that I met throughout these past four years, and who had an impact on my life.

I gratefully acknowledge the financial support received for the work performed during this Ph.D. thesis from the IWT (agentschap voor Innovatie door Wetenschap en Technologie), and by the First-MM project (Flexible Skill Acquisition and Intuitive Robot Tasking for Mobile Manipulation in the Real World) funded by the European Community's 7th Framework Programme, grant agreement First-MM-248258.

I would also want to thank all the people at IST Lisbon that I interacted with during our collaboration in the First-MM project. Your input, suggestions and contributions were invaluable, and many of the papers and research presented in this thesis would not have been the same without you. I especially want to thank Plinio, who helped a lot to get the iCub robot to do what we wanted it to do in our experiments.

Finally, I would like to thank my parents for supporting and helping me throughout all the study years, for always encouraging me to do what I was passionate about, and follow my dreams.

Bogdan Moldovan
Leuven, March 2015

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robotics is concerned with developing physical agents that can perform tasks by manipulating the physical world around them [Russell and Norvig, 2010]. Their development has been initially led by the goal of increasing productivity of manufacturing processes and taking place of humans in dangerous or hard to reach environments. Robots have been around in an industrial setting for more than fifty years, since 1961 when the Unimate robot was installed in a General Motors assembly plant [Koren and Koren, 1985]. Most industrial robots are mechanical handling devices, using a task program defined by a user in order to execute a clearly specified job [Koren and Koren, 1985]. However, these industrial settings are tightly controlled environments, which present very little uncertainty, and where the robots are used to perform repetitive tasks with very high accuracies.

On the other hand, real-world environments are unstructured and contain a high degree of uncertainty. But, as technology improved, there was a push to design and develop robots that can solve various tasks in real-world environments. The last decade has seen an increase in the number of domestic robots designed for tackling specific tasks in household environments. Some of the tasks these robots are used for include vacuum-cleaning, such as the Roomba developed by iRobot, floor washing or lawn-mowing. Other applications include the development of autonomous cars, such as the robotic vehicle Stanley that won the DARPA Grand Challenge in 2005 for autonomously navigating through an unknown off-road terrain [Thrun et al., 2006], and that of unmanned aerial vehicles (UAVs). Finally, the last few years has also seen an increase in research on humanoid robots, whose body is built to resemble a human body, and among which the most notable robots include the PR2, NAO and iCub.

For tackling complex tasks in real-world environments, robots are equipped with both sensors, such as cameras, laser-scanners, etc., allowing them to perceive their environment, and effectors, such as joints, grippers or legs, through which they can assert physical forces on the environment [Russell and Norvig, 2010]. The robots need to be able to deal with the noise associated with both their sensors and effectors, as well as to model in general the *uncertainty* in their physical world. This includes among others interpreting data from noisy sensors, processing image streams from cameras, or controlling noisy physical actuators for manipulation. To model this uncertainty, the use of probabilistic techniques is widespread in robotics, such as presented by [Thrun et al., 2005].

Secondly, the physical agents need to deal with *higher level knowledge* for reasoning and planning in their environment. For this purpose, early approaches, such as the well-known Shakey robot, used the logical STRIPS representation [Fikes and Nilsson, 1971]. Several other such symbolic representations [Hertzberg and Chatila, 2008, Stulp and Beetz, 2008] have later been introduced, yet most of these abstract away the noise and uncertainty from a full real-world robotics application.

To achieve the goal of modelling both the uncertainty and higher level knowledge needed for robotics applications, one can use *statistical relational learning* (SRL) [Getoor and Taskar, 2007, De Raedt and Kersting, 2008, De Raedt, 2008], which combines logical representations, probabilistic reasoning and machine learning, or more generally *probabilistic programming languages* (PPLs) [De Raedt et al., 2007, Goodman et al., 2008]. Recent works have started to investigate the use of SRL, and have shown how it can be used effectively to combine probabilistic and logical methods in robotics domains, such as in a kitchen scenario [Jain et al., 2009].

Lately, a promising approach for the development of humanoid robots' skills has been the learning of *object affordances*. Affordances model the robot-world interaction by capturing *action opportunities* to structure the robot's environment. They define what the robot can do with an object given its sensing and motor capabilities (e.g., a cup is handled in a different way than a ball). For this purpose, they model the relations between three variables: object properties, actions, and effects [Lopes et al., 2007, Montesano et al., 2008]. Affordances have been used in various settings, from the modelling of low-level relations between the numerical values of the object properties, actions, and effects as in [Montesano et al., 2008], to the high-level modelling of the effects of tool use, as in [Stoytchev, 2005, Sinapov and Stoytchev, 2007]. The concept of affordances follows the robotics developmental framework, which proposes to acquire new skills on top of old ones by experimentation and interaction with the environment [Lungarella et al., 2003].

Previous research involving affordances mostly considered isolated objects in the environment, for which the robot would learn an affordance model, which it would later use to solve a given task. This affordance model specifies for the robot how an object can be used, and modelling isolated objects is a reasonable assumption for several robotic tasks.

However, there are many other cases of real-life applications where it is necessary to consider the (spatial) relations between several objects, as well as relations in the environment, in order to correctly model object use. Thus affordance models in these scenarios should take these relations into consideration.

Consider for example an agenda, illustrated in various settings in Figure 1.1. The agenda in Figure 1.1a affords opening, and assuming that the robot is equipped with an optical character recognition system, text parsing. In Figure 1.1b, a cup is placed on top of the same agenda. Given this spatial relation involving the agenda and the cup, the agenda does not afford any of the two actions anymore. So modelling (spatial) relations between objects in the environment is important for determining actions afforded by an object. Finally, in the example in Figure 1.1c, there is no cup on top of the agenda, but the light is off in the room. In this case, the agenda still affords opening, but it no longer affords text parsing. So knowledge about the environment is also important for a more accurate modelling of affordances.

(a) Agenda on table (b) Agenda with cup on top (c) Agenda in low light

Figure 1.1: An agenda with various affordances due to different settings.

To handle situations similar to the ones in Figure 1.1, where relations between objects and knowledge about the environment need to be modelled, we introduce the concept of relational affordances. Relational affordances are an extension of the concept of affordances to the relational domain with the help of SRL. In our approach, the robot learns a relational affordance model in settings with two objects with different spatial relations between them, and then by generalising over objects it can employ the model in situations with an arbitrary number of objects.

We will use relational affordances to tackle several different robotics applications,

including action prediction in a table-top setting, occluded object search, or a shelf object arrangement planning scenario. We will thus show the feasibility of our approach for settings where spatial relations between objects and knowledge about the environment need to be taken into account.

1.1 Contributions

Single object affordance models have been used often in previous research for robot control. However, in real-world scenarios, objects interact with each other, and actions on an object might depend on its relations with other objects in the environment (e.g., an object might move if a nearby object is manipulated; a fork affords different actions if on the table or on the ground). **The overall research question we thus want to address within this thesis is the extension of affordances to a relational domain**, in order to properly model these situations, and thus extend the capabilities of robots to tackle these more complex real-world scenarios. Therefore, one of the main goals and contributions of this thesis is:

- **Learning relational affordance models**, for use in multi-object scenes by robustly going from two-object interactions to a variable number of objects by generalization over objects, using (background) logical rules, and modeling of probabilistic aspects.

We achieve this by using statistical relational learning techniques. Relational affordances allow modelling of actions and effects based also on the environment and relationships with other objects, and modelling of action effects on other objects in the environment with which the object acted upon might interact. As opposed to the previous methods of modelling affordances with BNs, the use of an SRL model works for any number of objects in the scene, while also providing increased model comprehensibility. We show how the relational affordance model can be learnt, in a discrete or continuous setting, from data collected during a behavioural babbling stage where the robot manipulates objects in its environment.

The second goal is to show through applications how the use of relational affordances can help robots solve various tasks in household environments. The use of relational affordances brought contributions to the following problems:

- **Models for two-arm robots**. Using SRL methods we can create a relational affordance model for two-arm actions, for settings where these can be approximated by a combination of the two single-arm actions

composing them. The arms may act simultaneously or sequentially, and the robot is given background knowledge about possible actions in its environment. Few studies have investigated two-arm manipulation, and SRL is used here to generalise and build a higher-level model for a set of two-arm actions settings in a household environment.

- **Planning action sequences.** We use a relational affordance model in order to define a state transition model in a table-top setting, and provide a planning algorithm to be used to infer the next best possible action for the robot to execute towards reaching the given goal. The goal presented to the robot is a high-level goal composed of spatial relations between the objects (e.g., place a cup near the plate to the right of the glass), which is in accordance with human-robot interaction approaches and represents a more realistic goal a human would ask from a robot.

- **Occluded object search**. Previously, searching for occluded objects focused on finding a specific given hidden object. Using the concept of relational affordances we can search for any object affording a given action. Multiple object types can afford the action, and each type allows for many different objects with size and shape modelled by probability distributions, thus relaxing some of the previous assumptions. Moreover, we allow for stacked objects, a more realistic modelling of objects in shelves, introducing more complex object spatial relations.

The contributions will thus be introduced and presented in a logical sequence. First we show how relational affordances can be learnt and used in a discrete setting and with a robot using only one of its arms with predefined actions. Then, we introduce the learning of relational affordances in a continuous setting, and we extend them for two-arm robots. Further, we show how the action repertoire of the robot can be extended by parameterising the actions. And finally, we will tackle a planning task involving a sequence of actions. Object search is related to table-top manipulation tasks, as the robot needs to first find the objects it needs to manipulate for its tasks.

We will illustrate the use of relational affordances in all the above mentioned areas, both in simulation settings (iCub and PR2 robot) and with a real robot (iCub robot). The experiments performed with these robots will show the relational affordances approach to be valid.

The main contribution of this thesis is the introduction of relational affordance models, which allows us to model object interactions in the environment in addition to being able to model the action opportunities of objects as previous affordance models do. As another contribution, we introduce a way of learning and using a probabilistic logic program that represents a relational affordance

model of the robot's world from data collected by the robot while exploring its environment. By using this approach, we are able to model interactions between pairs of objects, which has not been considered before in research in the area of affordances. This allows affordance models to be used in environments where objects interact with each other, and where the (spatial) relations between objects is important. Furthermore, we are also able to generalise over the number of objects in the environment, and have a model that can be more easily adapted to other settings. Another contribution of this approach is providing a model that integrates many of the elements needed to reason about the world the robot needs to act in. Firstly, the model is built from data collected from the robot's perception sensors, and its motor actions, taking the physical aspects of the robot into consideration. Secondly, we model the symbolic aspects of the world representation. Finally, the model incorporates both discrete and continuous data, which is necessary for a realistic and more general modelling of the environment. All these aspects will be described in more detail in the following chapters.

1.2 Thesis Roadmap

Chapter 2 presents background information on Bayesian Networks, logic programming, probabilistic programming languages, and planning.

Chapter 3 gives a brief overview of existing work in the field of affordances and relational robotics, and introduces the affordance context we will use for our tasks.

Chapter 4 introduces the core concept of relational affordances. Affordances, which define the action possibilities by the robot on an object in the environment, have focused on models for just one object. Several techniques have been used to model single object affordances, e.g., learning and using Bayesian Networks (BNs). Real-world applications involve scenarios with configurations of multiple objects interacting with each other, and where actions that can be performed on an object depend on its relations to other objects in the environment (e.g., a fork affords different actions if on the table or on the ground). In this chapter we employ recent advances in statistical relational learning to learn affordance models in such cases. The robot learns from a babbling stage of two-object interactions and the model is generalised to situations with an arbitrary numbers of objects. As opposed to the previous methods of modelling affordances with BNs, the use of SRL model works for any number of objects in the scene, while also providing increased model comprehensibility.

The chapter consists of research previously published in the following papers:

B. Moldovan, P. Moreno, M. van Otterlo, J. Santos-Victor, L. De Raedt. *Learning Relational Affordance Models for Robots in Multi-Object Manipulation Tasks*, in Proceedings of the 29th IEEE International Conference on Robotics and Automation (ICRA), St. Paul, MN, USA, 2012

B. Moldovan, P. Moreno, M. van Otterlo, J. Santos-Victor, L. De Raedt. *Statistical Relational Learning of Object Affordances for Robotic Manipulation*, in Latest Advances in Inductive Logic Programming, 2012

The next three chapters focus on applications of relational affordances.

Chapter 5 first introduces learning affordance models in a continuous domain setting, and then presents the modelling of complex manipulation tasks involving two-arm robots achieved with the help of the learnt relational affordances. Humanoid robots (*e.g.*, PR2, iCub, NAO, etc.) need to be able to manipulate objects in their environment, and one of their main characteristics is that they have two symmetrical arms for manipulation. A relational affordance model can first be learnt for one arm through a behavioural babbling stage, and then with the use of statistical relational learning, after constructing a symmetrical model for the other arm, two-arm manipulation actions can be modelled, where the arms can act sequentially or simultaneously. Furthermore, background knowledge about the environment and objects to be manipulated should be considered (*e.g.*, humans are more likely to manipulate two interacting objects, or parts of the same object, when using both hands). This can further be modelled with the help of logical rules, thus building an overall model for two-arm relational affordances in a multiple object environment, for settings where the two-arm action effects are a combination of the single-arm ones.

The material in this chapter has been based on the following article:

B. Moldovan, L. De Raedt. *Learning Relational Affordance Models for Two-Arm Robots*, in Proceedings of the 27th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Chicago, IL, USA, 2014

Chapter 6 presents a more complex planning manipulation task consisting of a sequence of actions, that is achieved with the help of the learnt relational affordances. We show how two-arm relational affordances can be used in a planning setting with high-level goals which are specified by (spatial) relations between the objects. In these cases, the robot can achieve its tasks through a sequence of its basic actions for which it has learnt affordance models, and needs to infer which action(s) to execute towards reaching the goal. Solving

this complex task goes towards a full table-top manipulation scenario with multiple interacting objects, where the robot can place any object where the user requests.

The material in this chapter has been based on the following articles:

> B. Moldovan, P. Moreno, M. van Otterlo. *On the Use of Probabilistic Relational Affordance Models for Sequential Manipulation Tasks in Robotics*, in Proceedings of the 30th IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany, 2013
>
> B. Moldovan, P. Moreno, D. Nitti, J. Santos-Victor, L. De Raedt. *Using Relational Affordances for Multiple-Action Two-Arm Manipulation Tasks*, to be submitted to the Robotics and Autonomous Systems journal before the preliminary defence.

Chapter 7 uses the concept of relational affordances to improve occluded object search performance. Before robots can tackle the complex manipulation tasks introduced before, one of the main problem they face is finding the objects they need to manipulate. However, in a real world environment, especially indoors, most objects are not immediately visible, but lie behind other objects. In this chapter we show how by learning and using a relational affordance model we can search for any of the multiple objects that afford a given action, each object type having a probability distribution over possible sizes and shapes, and where spatial relations between objects, such as co-occurrence and stacking, are modelled.

This chapter is based on the following paper:

> B. Moldovan, L. De Raedt. *Occluded Object Search by Relational Affordances*, in Proceedings of the 31st IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 2014

Chapter 8 is the concluding chapter, summarising the thesis and discussing opportunities for future work.

Some of the work performed during my PhD research has not been included in this thesis, as it is not related to the area of relational affordances investigated in this thesis. This included developing a Markov Chain Monte Carlo algorithm for estimating conditional probabilities based on sampling from an AND/OR tree for ProbLog, a general purpose probabilistic logic programming language [Moldovan et al., 2013b]. This was achieved with a parameterizable proposal distribution that generated the next sample in the Markov chain by

probabilistically traversing the AND/OR tree from its root, which holds the evidence, to the leaves. The second work not included in this thesis proposed a logical approach to predict the action of opening doors, together with the action point where the action should be performed, by learning a model and performing inference with the probabilistic programming language ProbLog [Moldovan et al., 2013a].

Chapter 2

Background

This chapter reviews a number of important concepts used throughout the thesis. Since in this thesis one of our aims is modelling the uncertainty in robotics settings, and we extend previous research using probabilistic models, we will start by reviewing Bayesian Network models in Section 2.1. Here we will also describe Linear Conditional Gaussian BNs, as well as parameter learning in a BN. Secondly, since our approach is concerned with using logical representations and extending affordances to a relational domain, we continue in Section 2.2 by reviewing basic concepts of logic programming, followed by introducing the basic concepts of probabilistic logic programming in Section 2.3. This latter section will also describe the syntax of the two probabilistic programming languages we will use in this thesis: ProbLog and Distributional Clauses. Finally, as one of the applications of relational affordances that we investigate tackles a planning task, we will finish this chapter by describing basic planning concepts in Section 2.4, including Markov Decision Processes (MDPs) and the STRIPS formalism.

2.1 Bayesian Networks

Bayesian Networks (BNs) [Pearl, 1988] are probabilistic graphical models that define a joint probability distribution over a finite set of random variables with finite domains in terms of conditional distributions for each variable given a subset of the others.

Formally, a BN is a directed graph $G = (X, E)$, where the vertices $X = \{X_1, ..., X_n\}$ represent the set of random variables, and E are the edges of the

graph. We call a node X_j a parent of a node X_i if $(X_j, X_i) \in E$. The set of all parents of a node X_i is denoted by $pa(X_i)$. A BN also defines a conditional probability distribution $P(X_i|pa(X_i))$ for each node $X_i \in X$.

The full joint distribution over all random variables of a BN is then given by the product of the individual distributions:

$$P(X) = \prod_{X_i \in X} P(X_i|pa(X_i)). \tag{2.1}$$

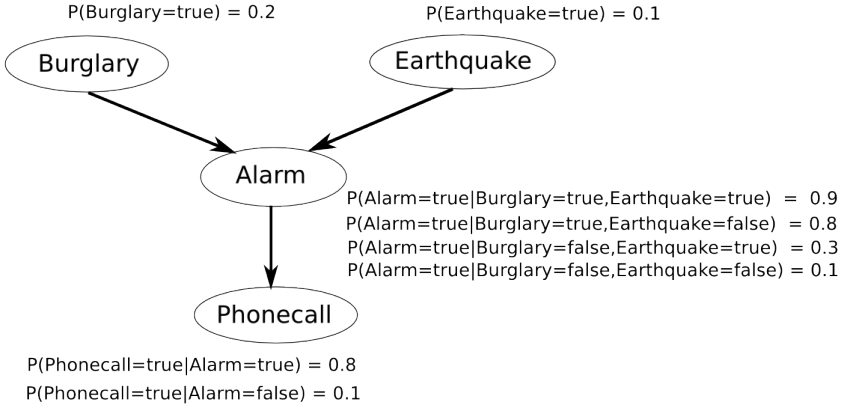


Figure 2.1: Alarm Bayesian Network.

Example 2.1. *Figure 2.1 presents a BN inspired by the well-known alarm BN presented in [Russell and Norvig, 2010] with Boolean random variables.*

It defines the joint distribution:
$P(Earthquake, Burglary, Alarm, Phonecall) = P(Earthquake) \cdot P(Burglary) \cdot P(Alarm|Earthquake, Burglary) \cdot P(Phonecall|Alarm)$.

For example, the probability:
$P(Earthquake = true, Burglary = false, Alarm = true, Phonecall = false) = 0.1 \cdot (1 - 0.2) \cdot 0.3 \cdot (1 - 0.8) = 0.0048$.

In our approach, BNs will be used in the initial modelling of the data collected by the exploratory actions of the robot in discrete settings. The use of BNs as part of creating our relational affordance models can be seen in Chapter 4.

2.1.1 Linear Conditional Gaussian BN

For modelling in a continuous domain, we will make use of *Linear Conditional Gaussian (LCG) Bayesian Networks* [Kjærulff and Madsen, 2005]. As opposed to the BNs introduced above where random variables had finite domains, an LCG BN specifies a distribution over a mixture of discrete and continuous variables. In an LCG, a discrete random variable may have only discrete parents, while a continuous random variable may have both discrete and continuous parents. A continuous random variable (X_c) will have a single Gaussian distribution function whose mean depends linearly on the state of its continuous parent variables (Y) for each configuration of its discrete parent variables (U). This LCG distribution can be represented as:

$$P(X_c = x_c|Y = y, U = u) = \mathcal{N}(x_c|M(u) + W(u)^T y, \sigma^2(u)), \qquad (2.2)$$

with $M(u)$ a table of mean values, $W(u)$ a table of regression (weight) coefficient vectors, and $\sigma(u)$ a table of variances (independent of Y). [Kjærulff and Madsen, 2005]

When representing LCG BNs graphically, we will represent discrete random variables by a single ellipse, and continuous ones by a double ellipse.

Consider the example LCG BN in Figure 2.2 which represents a simplified model of a robot manipulation task. The robot pushes an object on a table for some time, which is normally distributed with a mean of 2 seconds and standard deviation of 1. Round objects will have a higher speed due to lower friction, and also the uncertainty in their final displacement (in cm) will be higher.

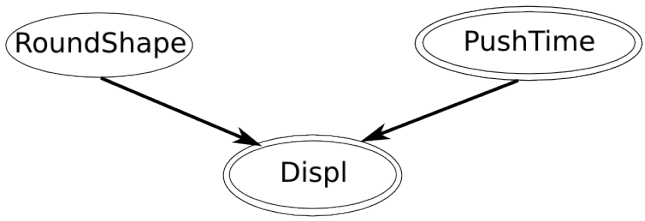


Figure 2.2: Manipulation example LCG BN.

We could define the probabilities of this LCG BN for example as:

$P(RoundShape = true) = 0.6 \qquad P(pushTime) \sim \mathcal{N}(2, 1)$
$P(Displ|PushTime, RoundShape = true) \sim \mathcal{N}(3 * pushTime + 0.8, 1.5)$
$P(Displ|PushTime, RoundShape = false) \sim \mathcal{N}(1.2 * pushTime + 0.3, 0.5)$

In our approach, LCG BNs will be used in the initial modelling of the data collected by the exploratory actions of the robot in continuous settings. Two of the robotics problems we investigate, namely creating two-arm actions models and multiple-action planning in Chapters 5 and 6 respectively, will show the use of LCG BNs.

2.1.2 Parameter Learning

For our approach, we need to learn the conditional probabilities in a Bayesian Network from data. Finding the numerical parameters for a probabilistic model is referred to as parameter learning. As in our settings, the robot observes during its exploration stage the complete data (i.e., there are values for every variable in the probability model), we make use of maximum-likelihood parameter learning, which we present below. In the case of incomplete data, if we need to learn the parameters of a model where some values are not observed, other algorithms such as Expectation-Maximisation (EM) can be used. For more extended details on the EM algorithm please refer to [Dempster et al., 1977].

We first present parameter learning in a BN. The maximum-likelihood parameter learning algorithm assumes it is provided with a dataset of n independent observations of all the variables X of the BN probability model: $D = \{d^1, ..., d^n\}$. Assume the BN where variable X_i has parents $pa(X_i)$ is defined by the conditional distributions parameters ϕ: $P(X_i|pa(X_i)) = \phi_{X_i|pa(X_i)}$. The goal of parameter learning is then to estimate the parameters of this network, ϕ, as close as possible to their true value. For maximum-likelihood, this is equivalent to finding ϕ that maximises $P(D|\phi)$, namely: $\arg\max_\phi P(D|\phi)$.

The likelihood function is given by:

$$L(D;\phi) = P(D|\phi) = \prod_{j=1}^{n} P(d^j|\phi) = \prod_{j=1}^{n} \prod_{i} P(X_i^j|pa(X_i^j), \phi). \tag{2.3}$$

In order to maximise this expression, it is more practical to maximise the logarithm of this function, or the log likelihood:

$$l(D;\phi) = \sum_{j=1}^{n} \sum_{i} P(X_i^j|pa(X_i^j), \phi). \tag{2.4}$$

By maximising the log likelihood, the obtained parameters will be:

$$\phi_{X_i|pa(X_i)} = \frac{n(X_i, pa(X_i))}{n(pa(X_i))}, \tag{2.5}$$

where $n(.)$ represents the counts for that specific configuration in the observed data.

So in effect, for a BN the maximum-likelihood parameter estimation method reduces to counting. Each parameter can be set by the relative count of the number of times the state exists in the dataset for a fixed joint parental state, compared to the other states with that parental configuration in the dataset.

Example 2.2. *Consider a simplified alarm network in a region with no earthquakes. We only have two nodes in the network, the alarm node and its parent, the burglary node. Suppose our learning dataset is composed of 10 independent observations. In six of them, there is no burglary and no alarm* $(Burglary = false, Alarm = false)$*. Out of the three where there is a burglary, the alarm sounds in two of them* $(Burglary = true, Alarm = true)$ *and in the other the alarm does not sound* $(Burglary = true, Alarm = false)$*. In one case the alarm sounds although there is no burglary* $(Burglary = false, Alarm = true)$*.*

The learnt parameters of the network by using maximum-likelihood will be:

$P(Burglary = true) = \frac{3}{10} = 0.3$*,* $P(Alarm = true|Burglary = true) = \frac{2}{3}$*, and* $P(Alarm = true|Burglary = false) = \frac{1}{7}$*.*

We will now present parameter learning in an LCG BN. Let us assume first a single continuous random variable node X_1 with no parents. This node defines the probability distribution:

$$P(X_1) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(X_1-\mu)^2}{2\sigma^2}}, \tag{2.6}$$

with parameters μ and σ. To learn the parameters given the dataset of n independent observations of the variable X_1: $D = \{d^1, ..., d^n\}$, we can compute, just like in the discrete case, the log likelihood function:

$$l(D; \mu, \sigma) == \sum_{j=1}^{n} log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(d^j-\mu)^2}{2\sigma^2}}. \tag{2.7}$$

To find the parameters μ and σ which maximise the log likelihood, we set the derivatives of $l(D; \mu, \sigma)$ with respect to μ and respectively σ to 0, and obtain:

$$\mu = \frac{\sum_{j=1}^{n} d^j}{n} \tag{2.8}$$

and

$$\sigma = \frac{\sum_{j=1}^{n} \sqrt{(d^j - \mu)^2}}{n}. \tag{2.9}$$

If we have a LCG BN with a continuous random variable node X_i with parents $pa(X_i)$, in a similar manner this time we need to maximise the conditional likelihood for the conditional distribution $P(X_i|pa(X_i))$:

$$P(X_i|pa(X_i)) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(X_i - \left(\sum_{k,X_k \in pa(X_i)} \mu_k X_k + \mu_c\right)\right)^2}{2\sigma^2}}, \tag{2.10}$$

in order to learn the parameters μ and σ.

The problem of maximising this log likelihood reduces to minimising $\left(X_i - \left(\sum_{k,X_k \in pa(X_i)} \mu_k X_k + \mu_c\right)\right)^2$. This problem of minimising of the sum of squared errors can be solved by standard linear regression, and algorithms have been developed to solve for the parameters μ, such as the one available in the BNT Toolbox for Matlab presented in [Murphy et al., 2001].

Using these approaches, we are able to learn the parameters of a BN or LCG BN by using maximum-likelihood parameter learning. Parameter learning for BNs and LCG BNs will be used in Chapters 4 to 6.

2.2 Logic Programming

In this section we will review the basic concepts of logic programming. For more details, please refer to [Lloyd, 1987] and [Flach, 1994]. We will illustrate these concepts on an example.

Example 2.3. *Consider the following example program that could be used by a robot that needs to look for an object of a required colour. All the objects are placed on one of several shelves.*

```
colour(book, green).
colour(cup, red).
colour(pen, blue).
shelf(1).
shelf(2).
onshelf(1, book).
onshelf(2, cup).
onshelf(1, pen).
findcolour(C) ← colour(Obj, C), onshelf(S, Obj), shelf(S).
```

Using Prolog notational conventions, *variable* names start with an uppercase letter, and *constants* start with lower case letters. In Example 2.3, the symbols

S, Obj and C denote variables, while the symbols book, cup, pen, green, red, blue, 1, and 2 are constants.

In Example 2.3, colour(book, green) is an atom, its intended meaning being that object book has colour green.

An atomic formula or atom $\texttt{pred}(\texttt{t}_1, ..., \texttt{t}_n)$ consists of a *predicate* $pred/n$ of arity n and t_i terms. A *term* is either a constant, a variable, or functor (function) $func/n$ applied on n terms. We can also have the negation of an atom, for instance not(onshelf(1, book)).

In Example 2.3, the following is a definite clause:

$$\texttt{findcolour}(\texttt{C}) \leftarrow \texttt{colour}(\texttt{Obj}, \texttt{C}), \texttt{onshelf}(\texttt{S}, \texttt{Obj}), \texttt{shelf}(\texttt{S}).$$

A *definite clause* is an expression of the form $\texttt{h} \leftarrow \texttt{b}_1, ..., \texttt{b}_n$, where h and b_i are atoms. It states that h is true whenever all b_i are true. All variables in clauses are implicitly universally quantified.

In Example 2.3, the following is a fact:

$$\texttt{shelf}(\texttt{1}).$$

A fact is a clause whose body is empty, thus consisting only of a single positive literal.

Example 2.3 is a logic program. A logic program is a finite set of definite clauses. Terms, atoms and clauses are *ground* if they do not contain any variables. For example, atom shelf(1) is ground, while atom shelf(S) is not.

A *substitution* $\theta = \{V_1/t_1, ..., V_n/t_n\}$ maps each variable V_i to a term t_i. Applying a substitution θ to an atom a yields $a\theta$, where each occurrence of V_i in a is replaced with t_i. For example, applying the substitution $\theta = \{\texttt{S/1}, \texttt{Obj/book}\}$ on the atom onshelf(S, Obj) results in: onshelf(1, book).

Two terms t_1 and t_2 are called *unifiable* if there exist substitutions θ_1 and θ_2 such that $t_1\theta_1 = t_2\theta_2$. A substitution θ is called the most general unifier of atoms a and b, denoted by $mgu(a, b)$ if and only if $a\theta = b\theta$, and for each substitution θ' such that $a\theta' = b\theta'$, there exists a substitution γ such that $\theta' = \theta\gamma$, with γ mapping at least one variable to a term other than itself.

Extending the substitution example previously mentioned, $\theta = \{\texttt{S/1}, \texttt{Obj/book}\}$ is a most general unifier of onshelf(S, Obj) and onshelf(1, book), but $\theta_1 = \{\texttt{S/1}, \texttt{Obj/GreenObj}, \texttt{GreenObj/book}\}$ is not.

The set of ground atoms that can be constructed using the predicates, functors and constants in a logic program is called the Herbrand base of the program. Subsets of the Herbrand base are called Herbrand interpretations. A Herbrand

interpretation is a model of a clause $\mathtt{h} \leftarrow \mathtt{b_1}, ..., \mathtt{b_n}$ if for every substitution θ where all $b_i\theta$ are in the interpretation then $h\theta$ is also in the interpretation. A Herbrand interpretation is a model of a logic program if it is a model for all clauses in that logic program. The least Herbrand model is the smallest such model, i.e., the minimal subset.

A logic program entails an atom b if and only if b is true in the least Herbrand model of the logic program. Inference in logic programming means deciding whether a *query* consisting of a conjunction of atoms: $? - \mathtt{q_1}, ..., \mathtt{q_n}$ is logically entailed by the program for some substitution θ. If this is the case, then it is said that the query succeeds. For extensive details on semantics of logic programs and inference please refer to [Lloyd, 1987] and [Flach, 1994].

Most logic programming systems, such as Prolog, use *SLD-resolution* as a feasible way of inferring whether a query holds. SLD-resolution is a refutation process, where the negation of the query is added to the program and resolution is used to derive the empty clause. To refute the query $? - \mathtt{q_1}, \mathtt{q_2}, ..., \mathtt{q_n}$, SLD-resolution selects from the logic program one clause $\mathtt{h} \leftarrow \mathtt{b_1}, ...\mathtt{b_n}$ such that $\theta = mgu(h, q_1)$ and computes the resolvent: $? - \mathtt{b_1}\theta, ...\mathtt{b_n}\theta, \mathtt{q_2}\theta, ..., \mathtt{q_n}\theta$.

This resolution step is then repeated. If the sequence of resolution steps reaches the empty query (denoted by $\square$), then the query $? - \mathtt{q_1}, \mathtt{q_2}, ..., \mathtt{q_n}$ is proven.

Example 2.4. *Consider the logic program in Example 2.3. The query* `findcolour(red)` *can be proved as follows using SLD-resolution:*

$? - \mathtt{findcolour(red)}$
$? - \mathtt{colour(Obj, red), onshelf(S, Obj), shelf(S)}$
$? - \mathtt{onshelf(S, cup), shelf(S)}$
$? - \mathtt{shelf(2)}$
$? - \square$

2.3 Probabilistic Logic Programming

The distribution semantics defined by [Sato, 1995] provides a formal basis for extending logic programming languages with probabilistic elements. The main difference to the logic programs introduced in Section 2.2 is the presence of a set of probabilistic facts, which are not always set to true, but whose truth value is probabilistic. We introduce here the basic ideas of distribution semantics, which will be useful when explaining the ProbLog and Distributional Clauses (DC) probabilistic programming languages. For more extensive details about distribution semantics please refer to [Sato, 1995].

A program can be defined by $F \bigcup BK$, where F is a set of unit clauses, called facts, and BK is a set of (possibly non-unit) clauses, called background knowledge. We assume that no fact in F unifies with the head of a rule in BK. Each ground instance of a probabilistic fact $p_i :: f_i \in F$ represents a random variable that is true with the given probability p_i. In fact, each ground f_i specifies an atomic choice, i.e., we can choose to include f_i as a fact in the program with probability p_i, or not (with probability $1 - p_i$). An assignment of truth values to the ground f_i in the program defines a possible world. The semantics of the program is then given by probability distributions over subsets of the facts F (called subprograms).

The distribution semantics is a generalisation of the least Herbrand model semantics. The distribution over the probabilistic facts of the program defines a distribution over least Herbrand models. Any such Herbrand model is a possible world. The success probability of a query then is equivalent with the probability that the query is true in a randomly sampled Herbrand model.

The distribution semantics can also be viewed as a possible worlds semantics, with ground atoms treated as random variables and worlds corresponding to interpretations assigning truth values to all ground atoms in the definite clause program.

A *total choice* [Poole, 1997] is an assignment of truth values for *all* random variables. To each total choice we can associate a probability. This is simply the product of the probabilities of the atoms chosen for inclusion in the total choice, as these random variables are marginally independent.

The distribution over total choices induces a probability distribution P over possible worlds, which also defines the success probability $P_s(q)$ of a query q as: $P_s(q) = P(\{w|q \text{ is true in the possible world } w\})$. Computing $P_s(q)$ is one of the main inference tasks which we will require when using a probabilistic programming language.

There are many Probabilistic Programming Language (PPL) formalisms, such as Church [Goodman et al., 2008], BLOG [Milch et al., 2005], IBAL [Pfeffer, 2001], and PRISM [Sato and Kameya, 1997]. In our approach, for ease of exposition, we will use ProbLog [De Raedt et al., 2007] for discrete domains, and Distributional Clauses (DCs) [Gutmann et al., 2011b] for continuous and temporal domains. Although DCs can also be used to model discrete domains, ProbLog has been chosen for ease of explanation to first introduce all the concepts (in a discrete setting). We will now introduce the syntax of these PPLs and illustrate them with examples.

2.3.1 ProbLog

We will briefly review the probabilistic programming language ProbLog. For more details about this PPL please consult [De Raedt et al., 2007]. We will illustrate all concepts on the following example.

Example 2.5. *Consider the following ProbLog program modelling a variation of the classical alarm problem presented in [Russell and Norvig, 2010]. The program has the random variables:* `burglary`*,* `earthquake`*,* `hears_alarm(john)` *and* `hears_alarm(mary)`*, and states that there is an alarm whenever there is a burglary or an earthquake. The last clause states that if there is an alarm and a person hears the alarm, that person will call.*

```
0.1 :: burglary.
0.2 :: earthquake.
0.7 :: hears_alarm(john).
0.6 :: hears_alarm(mary).
alarm ← burglary.
alarm ← earthquake.
calls(Pers) ← alarm, hears_alarm(Pers).
```

The probabilistic facts $p_i :: f_i \in F$ signify that $f_i\theta$ is true with probability p_i for all substitutions θ grounding f_i. The probabilistic facts in Example 2.5 are the following:

$$F = \{\texttt{0.1} :: \texttt{burglary}, \texttt{0.2} :: \texttt{earthquake}, \\ \texttt{0.7} :: \texttt{hears\_alarm(john)}, \texttt{0.6} :: \texttt{hears\_alarm(mary)}\}.$$

A ProbLog program $\mathcal{T}$ with $F = \{p_1 :: f_1, ..., p_n :: f_n\}$ defines a probability distribution over logic programs $L \subseteq L_T = \{f_1, ..., f_n\} \bigcup BK$ as:

$$P(L|\mathcal{T}) = \prod_{f_i \in L} p_i \prod_{c_i \in L_T \setminus L} (1 - p_i). \tag{2.11}$$

In terms of logic, L is a complete interpretation that states that all atoms contained in L are true, while all the rest are false.

For example, the probability of the total choice $\{\texttt{burglary}, \texttt{hears\_alarm(john)}\}$, expressing that a burglary and not an earthquake has occurred, while John heard the alarm but Mary had not, is: $0.1 \times (1 - 0.2) \times 0.7 \times (1 - 0.6) = 0.0224$.

To compute the success probability of a query (e.g., the query `alarm`) we need to compute the probability that the query is provable in a randomly sampled logic program. As there are exponentially many subprograms $L \subseteq L_T$ (e.g., in this case 2^4 subprograms), it is infeasible to enumerate all of them explicitly. Rather

than enumerating these explicitly, one would compute the proofs of the query and observe that `alarm` is true exactly when `earthquake` or `burglary` is true. Several more efficient inference methods have been proposed for computing the success probability of a query, both for computing the exact value and for computing an approximation. For more details on these, please refer to [De Raedt et al., 2007] or [Gutmann et al., 2011b] among others.

For example, using the ProbLog exact inference method, the (success) probability of the query `calls(mary)` for the ProbLog program in Example 2.5 is: $P_s(\texttt{calls(marry)}) = 0.168$.

Note that predicates in the body of clauses can also be used to define constraints in the program. A predicate b_i is a constraint for a predicate h if it is present in the body of all the clauses whose head is h. For example, in Example 2.5, `alarm` is a constraint for `calls(Pers)`: `alarm` needs to be true for `calls(Pers)` to be true.

For ease of presentation of programs, sometimes we will also use annotated disjunctions. For example, to model that the shape of an object a robot detects is randomly chosen from a set of two predefined shapes (i.e. the shape can take exclusively one of the two values with a probability of $\frac{1}{2}$) one can write the following clause:

$$\tfrac{1}{2} :: \texttt{shape(Obj, cube)}; \tfrac{1}{2} :: \texttt{shape(Obj, cylinder)} \leftarrow \texttt{obj(Obj)}.$$

where variable `Obj` is universally quantified over the set of all objects.

Formally, an annotated disjunction, which is a generalisation of a probabilistic fact, is a statement $p_1 :: f_1; ...; p_n :: f_n \leftarrow b$, where the body b is a conjunction of atoms. For all substitutions θ grounding $b\theta$ and all $f_1\theta, ..., f_n\theta$, when $b\theta$ is true at most one $f_i\theta$ is true; this $f_i\theta$ becomes true with probability p_i [Vennekens et al., 2009].

We can now make the link to the BNs introduced in Section 2.1, and we can show how any BN can be modelled with the help of ProbLog and annotated disjunctions. The procedure, which we will summarise here, is explained in more detail in [Vennekens et al., 2009].

Consider any node X_i of the BN, with domain $\{x_i^1, ..., x_i^k\}$. Assume node X_i has m parents $pa(X_i) = \{X_{p_1}, ..., X_{p_m}\}$. Let $f_i(x_i^j)$ be a probabilistic fact that we use to denote that some random variable X_i takes on value x_i^j. Assume that when $pa(X_i)$ take on values $w_1, ..., w_m$, we have the conditional probability table giving probabilities $p_i^j = P(X_i = x_i^j | X_{p_1} = w_1, ..., X_{p_m} = w_m)$. Then, we can write the annotated disjunction clause: $(p_i^1 :: f_i(x_i^1); ...; p_i^k :: f_i(x_i^k) \leftarrow f_{p_1}(w_1), ..., f_{p_m}(w_m)$. The BN can be represented by the program consisting

of the set of all such clauses for all nodes X_i and for all parent values in their domain.

Example 2.6. *The BN from Figure 2.1 can be represented by the following ProbLog program using annotated disjunctions:*

```
0.2 :: burglary(true); 0.8 :: burglary(false) ← true.
0.1 :: earthquake(true); 0.8 :: earthquake(false) ← true.
0.9 :: alarm(true); 0.1 :: alarm(false) ← burglary(true),
   earthquake(true).
0.8 :: alarm(true); 0.2 :: alarm(false) ← burglary(true),
   earthquake(false).
0.3 :: alarm(true); 0.7 :: alarm(false) ← burglary(false),
   earthquake(true).
0.1 :: alarm(true); 0.9 :: alarm(false) ← burglary(false),
   earthquake(false).
0.8 :: phonecall(true); 0.2 :: phonecall(false) ← alarm(true).
0.1 :: phonecall(true); 0.9 :: phonecall(false) ← alarm(false).
```

ProbLog also supports parameter learning through *learning from interpretations* (LFI) [Gutmann et al., 2011a]. LFI uses a ProbLog program $T(p)$ for the unknown parameters $p = \langle p_1, ..., p_n \rangle$, and our gathered training data $D = \{d^1, ..., d^M\}$, d^i the data from instance i, to compute the maximum likelihood parameter estimation: $\hat{p} = \arg\max_P P(D|T(p)) = \arg\max_P \prod_{m=1}^{M} P(d^m|T(p))$, thus obtaining the probability parameters of the (relationally encoded) BN.

ProbLog will be used for the modelling of relational affordances in a discrete setting. This will be done in Chapter 4 of the thesis.

2.3.2 Distributional Clauses

After introducing the main concepts of PPLs with the help of ProbLog, we introduce Distributional Clauses (DCs), which will be used for continuous domains models. For more extensive details on DCs, please consult [Gutmann et al., 2011b] and [Nitti et al., 2013].

Formally, a DC is an expression of the form $h \sim \mathcal{D} \leftarrow b_1, ..., b_n$, where b_i are atoms and $\sim$ a binary predicate written in infix notation. Informally speaking, whenever the conditions in the body $b_1, \ldots, b_n$ hold, a random variable h is defined with distribution $\mathcal{D}$. A DC is a powerful template to define conditional probabilities, indeed b_i, h, and $\mathcal{D}$ can contain logical variables that parameterise the clause. Formally, in a DC, each ground instance of the clause

$(h \sim \mathcal{D} \leftarrow b_1, ..., b_n)\theta$, for a substitution θ, defines the random variable $h\theta$ being distributed according to distribution $\mathcal{D}\theta$ when all $b_i\theta$ hold.

For example, we can model that all objects pushed by the robot have a displacement represented by a Gaussian distribution with mean $5cm$ and covariance $1cm$:

$$\texttt{displ}(\texttt{Obj}) \sim \texttt{gaussian}(5, 1) \leftarrow \texttt{push}(\texttt{Obj}).$$

The term $\mathcal{D}$ that represents the distribution can be nonground: values, probabilities, or distribution parameters can be related to conditions in the body. Additionally, a term $\simeq(d)$ constructed from the reserved functor $\simeq/1$ represents the value of the random variable d.

For example, given two objects pushed as above, we can model a predicate that is true whenever the first object `Obj1` moved farther than the second one `Obj2` as follows:

$$\texttt{movedfarther}(\texttt{Obj1}, \texttt{Obj2}) \leftarrow \simeq(\texttt{displ}(\texttt{Obj1})) > \simeq(\texttt{displ}(\texttt{Obj2})).$$

Several inference algorithms have been proposed for the DC distribution semantics for computing the success probability of a query q: $P_s(q)$. The inference algorithm proposed in [Gutmann et al., 2011b] uses sampling, and magic sets [Bancilhon and Ramakrishnan, 1986] to generate only facts relevant to the query. The sampling algorithm proposed by [Nitti et al., 2013] is based on backward reasoning. For more details on the specifics of these inference algorithms please refer to [Gutmann et al., 2011b] and [Nitti et al., 2013].

For example, given the example displacement clauses defined above, one can compute the probability of the displacement of a specific pushed object, for example a book, being greater than $6cm$: $P(\simeq(\texttt{displ}(\texttt{book})) > 6) = 0.158$.

We will also use DC-style syntax to define the state of the robot's environment at a given time. A state will consist of a conjunction of grounded terms. For example, after the mentioned push action, a state can consist of the book object which has the shape of a prism, and the displacement of $6cm$:

$$\simeq(\texttt{shape}(\texttt{book})) = \texttt{prism}, \simeq(\texttt{displ}(\texttt{book})) = 6.$$

For convenience, where this use is unambiguous, we will sometimes drop the $\simeq$ functor notation, so the state above can also be written as:

$$\texttt{shape}(\texttt{book}) = \texttt{prism}, \texttt{displ}(\texttt{book}) = 6.$$

DCs will be used for the modelling of relational affordances for continuous settings. DCs will be used for modelling affordances for all the three robotics applications that we showcase, in Chapters 5 through 7.

2.3.3 Dynamic Distributional Clauses

Dynamic Distributional Clauses (DDCs) [Nitti et al., 2013] are an extension of Distributional Clauses for temporal domains. It defines a discrete-time stochastic process following the same idea of a Dynamic Bayesian Network [Murphy, 2002]. We need sets of clauses that define:

1. the prior distribution: $\{\mathtt{h}_0 \sim \mathcal{D} \leftarrow \mathtt{body}_0\}$,
2. the state transition model: $\{\mathtt{h}_{t+1} \sim \mathcal{D} \leftarrow \mathtt{body}_t\}$,
3. the measurement model: $\{\mathtt{z}_{t+1} \sim \mathcal{D} \leftarrow \mathtt{body}_{t+1}\}$,
4. a random variable at time t from other variables at the same time: $\{\mathtt{h}_t \sim \mathcal{D} \leftarrow \mathtt{body}_t\}$.

For example, to model an environment with balls, where the next position of every ball is equal to the current position plus some Gaussian noise, one can write:

$$\mathtt{pos(Obj)}_{t+1} \sim \mathtt{gaussian}(\simeq(\mathtt{pos(Obj)}_t), \mathtt{cov}) \leftarrow \mathtt{ball(Obj)}.$$

This signifies that the position of the object at time $t+1$ is given by a Gaussian distribution with mean equal to the position of the object at time t, and covariance cov.

This state transition model can be used for defining a planning setting as we will later show in Section 2.4.2. With the help of DDCs, we can define the state of the robot's environment at different times. For example, at time t, the state consisting of a blue ball at position $3cm$ is:

$$\mathtt{ball(blueball)}, \simeq(\mathtt{pos(blueball)}_t) = 3,$$

while at time $t+1$ the state given by the same ball at position $4cm$ is:

$$\mathtt{ball(blueball)}, \simeq(\mathtt{pos(blueball)}_t) = 4.$$

Given a set of DDC clauses, Distributional Clauses Particle Filter (DCPF) [Nitti et al., 2013] can be used to perform filtering inference. Formally, filtering or state estimation computes the probability density function $P(x_t|z_{1:t}, a_{1:t})$, where x_t is the current state, $z_{1:t}$ is the set of observations, and $a_{1:t}$ the actions (inputs) performed from time step 1 to t.

Note that if no DDC clauses that define the random variable $\mathtt{h}_{t+1}$ apply to a given state, $\mathtt{h}_{t+1}$ is undefined. This is useful to describe negative effects, e.g. a

fact that is no longer true[1], or when a random variable is no longer needed in the next state.

For example, if we care only about the objects on a table for manipulation, we can ignore and thus remove from the next state all the objects that fall off the table or that are grabbed by a human. All the random variables related to those objects will automatically be removed. In our ball example, if `ball(blueball)` is removed, all random variables (facts) `pos(blueball)` are implicitly removed from the next state.

This property allows us to use DDCs to define add and delete lists in a STRIPS-planning setting. We will show later in Section 2.4.2 how we can represent a STRIPS operator with DDCs.

We will use DDCs for modelling relational affordances in temporal domains, such as modelling the multiple-action planning tasks in Chapter 6.

2.4 Planning

Probabilistic planning is generally performed by solving a Markov decision process (MDP) for fully-observable problems, or a partially observable Markov decision process (POMDP) for partially-observable problems. We will start this section by introducing MDPs. To describe the planning problems themselves, one of the first languages used was the STRIPS [Fikes and Nilsson, 1971] language, which is the base for many languages used to express planning problems developed since then. We will introduce STRIPS in the second part of this section.

2.4.1 Markov Decision Processes

An MDP [Wiering and van Otterlo, 2012, Sutton and Barto, 1998] consists of a set S of states (with an initial state s_0), a set A of actions the agent can take in each state, a state transition model $T : S \times A \times S \rightarrow [0, 1]$ where $T(s_t, a_t, s_{t+1}) = P(s_{t+1}|s_t, a_t)$, and a reward function $R : S \times A \rightarrow \mathbb{R}$ that assigns a reward $r_t(s_t, a_t)$ when the agent is at state s_t and performs action a_t. In our approach we use relational states, that is, a state s is a set of ground relational atoms and/or pairs (variable, value). The goal of the agent is to maximize the expected (discounted) reward $\mathcal{E}[\sum_{t=0}^{\delta} \gamma^t r_t]$. If δ is finite we have a finite horizon MDP, otherwise we have an infinite horizon MDP. The term

[1] We assume the closed-world assumption: anything that is undefined is considered false.

$\gamma \in [0, 1]$ is called a discount factor, and it is needed to keep the sum bounded for infinite horizon MDPs. For finite horizon MDPs, a value $\gamma = 1$ is often used.

To maximize the expected reward, we need to find a (deterministic) policy π that assigns, for each state s and time t, the action to perform. Without loss of generality, for infinite horizon MDPs we can use stationary policies $\pi : S \to A$ that do not depend on the time t. Using a stochastic policy $\pi : S \times A \to [0, 1]$ assigns a distribution over actions $P(a|s)$ for each state s.

The expected reward starting from state $S_t = s$ and following a policy π is called the value function (or V function). For finite horizon MDPs (assuming $\gamma = 1$) it is:

$$\mathcal{V}^{\pi}_{\delta}(s) = \mathcal{E}[\sum_{k=0}^{\delta} r_{t+k} | S_t = s, \pi], \tag{2.12}$$

and for infinite horizon MDPs:

$$\mathcal{V}^{\pi}(s) = \mathcal{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | S_t = s, \pi]. \tag{2.13}$$

In the finite case the V function depends on the number δ of remaining steps, which is not the case for infinite horizon MDPs.

The expected reward starting from state $S_t = s$ executing action $A_t = a$ and following a policy π is called the action-value function (or Q function). For finite horizon MDPs:

$$\mathcal{Q}^{\pi}_{\delta}(s, a) = \mathcal{E}[\sum_{k=0}^{\delta} r_{t+k} | S_t = s, A_t = a, \pi], \tag{2.14}$$

and for infinite horizon MDPs:

$$\mathcal{Q}^{\pi}(s, a) = \mathcal{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | S_t = s, A_t = a, \pi]. \tag{2.15}$$

An optimal policy π^* is a policy that maximises the V function for all states. In general, the goal when solving a planning problem is to obtain such a policy.

2.4.2 STRIPS

The STRIPS (Stanford Research Institute Problem Solver) language was introduced by Richard Fikes and Nils Nilsson in [Fikes and Nilsson, 1971] and is a formal language used to describe planning problems. This language is at

the base of many derived planning languages now in use, such as the Planning Domain Definition Language (PDDL) [Mcdermott et al., 1998] which is an ongoing attempt at standardising planning languages. As we will use STRIPS notation in our approach when it relates to planning tasks, we will briefly introduce it below.

In STRIPS a world model is described through the definition of an initial state, a set of goal states, and (action) operators. States are represented by conjunctions of function-free ground literals. Usually the closed-world assumption is used, meaning that any literal not mentioned in the state description is assumed to be false.

Example 2.7. *Consider the following setting with two adjacent rooms: room*1 *and room*2*. The robot is in room*1 *and its task is to go to a table that is found in room*2*.*

The initial state of the system can be described by:

$$\texttt{robotin(room1)} \wedge \texttt{tablein(room2)} \wedge \texttt{nextto(room1, room2)}$$

Goals are described by a conjunction of literals. These do not need to be ground (i.e., can contain variables). For example, in the setting from Example 2.7, the goal of the robot can be to reach a room where there is a table:

$$\texttt{robotin(R)} \wedge \texttt{tablein(R)}$$

A STRIPS operator, which is used for defining an action, is a four-tuple: $< a, pre, add\_list, del\_list >$. The action description is a. The precondition pre of the action defines what conditions must be true in order for the action to be applicable, and it is represented by a conjunction of atoms. All variables present are assumed to be universally quantified. The action a will be applicable in a state s if there is a way to instantiate the variables in pre such that it is true in state s. The effects (or postconditions) of applying the action operator are given by an add list add_list and a delete list del_list, all defined by a conjunction of literals. The add list specifies which literals become true due to the action and hold in the resulting next state. The delete list specifies which literals are no longer true and must be deleted. STRIPS also makes the frame assumption, namely that any literal not mentioned in the add and delete lists remains unchanged.

Example 2.8. *For example, in the setting from Example 2.7, we can define the four-tuple operator for the action move for the robot, which causes the robot to move from one room to another adjacent room:*

$$< \texttt{move(X, Y)}, \texttt{robotin(X)} \wedge \texttt{nextto(X, Y)}, \texttt{robotin(Y)}, \texttt{robotin(X)} >$$

A plan will then consist of a sequence of operators that can be executed from the initial state and that leads to a goal state.

We will mostly use a probabilistic STRIPS formalism, such as the one introduced in [Zettlemoyer et al., 2005, Pasula et al., 2004], where in comparison to classical STRIPS, each action has a small number of outcomes, each associated with a probability that it might occur *prob*. In this case, an action operator will be a five-tuple $< a, pre, add\_list, del\_list, prob >$,

Example 2.9. *For example, assume that in the setting from Example 2.7, the move action only has an* 80% *probability of success, while there is a* 20% *chance that the robot will remain in the same room due to errors in its navigation. The action representation can then be defined by:*

$$< \texttt{move(X,Y)}, \texttt{robotin(X)} \wedge \texttt{nextto(X,Y)}, \texttt{robotin(Y)}, \texttt{robotin(X)}, 0.8 >$$
$$< \texttt{move(X,Y)}, \texttt{robotin(X)} \wedge \texttt{nextto(X,Y)}, \varnothing, \varnothing, 0.2 >$$

This means that if, for example, action $\texttt{move(room1,room2)}$ *is executed in the initial state from Example 2.7, the robot can end up in one of the following two states:*

$$\texttt{robotin(room2)} \wedge \texttt{tablein(room2)} \wedge \texttt{nextto(room1,room2)}$$

with probability 0.8*, or*

$$\texttt{robotin(room1)} \wedge \texttt{tablein(room2)} \wedge \texttt{nextto(room1,room2)}$$

with probability 0.2.

We have seen in Section 2.3.3 how DDCs can be used to model a state transition. We will now show here how we can represent the state transition defined by a probabilistic STRIPS operator with the help of DDCs.

Example 2.10. *Consider the probabilistic STRIPS operator defined in Example 2.9. We will show how this setting can be represented with the help of DDCs. The* `robotroom` *random variable models the room the robot is in.*

To model that in the initial state the robot can be in any of the two rooms with equal probability:

$$\texttt{robotroom}_0 \sim \texttt{uniform}([\texttt{room1}, \texttt{room2}]) \leftarrow \texttt{true}.$$

The state transition in Example 2.9 can be modelled with DDCs as follows:

$$\texttt{robotroom}_{t+1} \sim \texttt{finite}([0.8 : \texttt{Y}, 0.2 : \texttt{X}]) \leftarrow \simeq(\texttt{robotroom}_t) = \texttt{X},$$
$$\texttt{nextto(X,Y)}, \texttt{move(X,Y)}_t.$$

In this DDC clause, $\texttt{move}(\texttt{X},\texttt{Y})_{\texttt{t}}$ is the action executed at time t, and $\texttt{nextto}(\texttt{X},\texttt{Y})$ holds if the rooms X and Y are adjacent.

We will refer to a probabilistic STRIPS representation represented with DDCs when tackling our multiple-object table-top planning task in Chapter 6.

Chapter 3

Affordances Overview and Relational Robotics

This chapter has three main goals. Firstly, it is to give a brief overview of the concept of affordances, introduce some of the formalisms used, and then present some of the various settings in which affordance models have been used in robotics research (Section 3.1). Secondly, in Section 3.2, we will briefly present some of the research that used relational learning and models in the field of robotics. Finally, the third goal is to present the affordance setting which this thesis is based on (Section 3.3), and based on this to present our relational affordances (Section 3.4).

The overall aim of this chapter is thus to place our current research on relational affordances and their applications in robotics in a historical perspective, and introduce the specific affordance setting which we will extend to a relational domain. The chapter is not meant as an exhaustive survey in these fields, but rather its purpose is to highlight some of the major developments and formalisms.

3.1 Affordances

We will first introduce the concept of affordances, and several of the more known formalisms used to define them. We closely follow [Şahin et al., 2007], work which can also be referred to for a wider survey on the field of affordances.

The concept of affordances was originally proposed by J. J. Gibson (1904–1979), an influential 20th century psychologist, in his book "The Ecological Approach to Visual Perception" [Gibson, 1979] which presented elements of ecological psychology. There he introduced the concept in order to refer to the action opportunities (or possibilities) offered to an organism by its environment, and postulated that the organism and its environment complement each other [Şahin et al., 2007]. For example, a ball might afford throw-ability, or a rigid surface affords walk-ability.

At this moment there is no universally accepted formal definition of the term, but one of the most frequently quoted definitions of the term by Gibson is the following:

"*The affordances of the environment are what it offers the animal, what it provides or furnishes, either for good or ill. The verb to afford is found in the dictionary, but the noun affordance is not. I have made it up. I mean by it something that refers to both the environment and the animal in a way that no existing term does. It implies the complementarity of the animal and the environment." [Gibson, 1979]*

Gibson proposed that affordances are directly perceivable by the organism, who would thus have clear knowledge of each object it observes in its environment. Gibson's view of a system composed of the organism and the environment together, including his introduction of affordances, has been one of the bases of ecological psychology [Şahin et al., 2007].

Apart from its introduction in ecological psychology, the concept of affordances has had an impact on other, seemingly unrelated fields. In computer science, the concept has been used in research in human-computer interaction, and lately in the area of autonomous robotics. An ecological approach to the design of robotic agents can take advantage of the relationships between the robot and its environment, and so the design of such a robot can focus more on the details of modeling the interaction with the environment rather than on constructing and maintaining complex internal representations, and thus being more flexible and better able to respond to dynamic, real world conditions [Horton et al., 2012].

Research has also shown that affordances, and how we perceive objects, have a deeper link to an animal's physiology. Studies have shown that canonical neurons, which are located in the ventral premotor cortex in the F5 area of a monkey, fire to the sight of an object of a certain shape or which has a certain function [Murata et al., 1997]. This neuron response to visualising the objects, and which does not require an action to be applied to the object, is linked to the understanding of the concept given by the object. Another set of neurons, called the mirror neurons, fire whenever an individual

performs and action, but also when that individual observes another individual performing the same action. This has a fundamental role in action understanding and imitation learning [Rizzolatti and Craighero, 2004]. In robotics, this link between affordances, and canonical and mirror neurons, has been noted in research such as [Sahin and Erdogan, 2009].

We will now proceed with a brief overview of affordances in the field of robotics.

3.1.1 Affordance Formalisms

In the field of robotics, the similarity of the arguments of Gibson's theory and the area of behaviour-based robotics has been noted. It was the same line of thinking applied to two different domains, ecological psychology and behaviour-based robotics [Şahin et al., 2007]. However, because Gibson's description of affordances was ambiguous, over time several attempts have been made to formalize the concept. We will briefly touch here on several of these that were introduced over the years.

One of the first formalisations of the concept of affordances was introduced in [Turvey, 1992]. In his formalism, an affordance is a disposition. These are defined in pairs, affordances as dispositions of the environment, which are properties of objects, and their complement which are the "effectivities" of the organism. When two complement dispositions meet in space and time, they get actualized. For example, "climbability" of stairs is their disposition, and the complement is a person's effectivity of climbing, and when they get actualised they result in the stairs being climbed.

The formal definition in [Turvey, 1992] is:

" *Let W_{pq} (e.g., a person-climbing-stairs system) $= j(X_p, Y_q)$ be composed of different things Y (person) and X (stairs). Let p be a property of X and q be a property of Y. Then p is said to be an affordance of X and q the effectivity of Y (i.e., the complement of p), if and only if there is a third property r such that:*

- *$W_{pq} = j(X_p, Y_q)$ possesses r [where $j(\cdot)$ is the juxtaposition function that joins X_p and Y_q].*
- *$W_{pq} = j(X_p, Y_q)$ possesses neither p nor q.*
- *Neither Y nor X possesses r.* "

Later a different formalism of affordances was proposed in [Chemero, 2003]. In Chemero's formalism, affordances are not properties of either the environment

or the organism, but rather relations between the abilities of organisms and features of the environment.

The affordance formalism presented in [Chemero, 2003] is the following:

" $Affords - \phi(environment, organism)$, *where* ϕ *is a behavior.*"

One example illustrating the difference in Chemero's formalism compared to the previous approaches is a relation such as: $Heavier\_than(robot, ball)$. This relation is not inherent in any of the two objects, but can exist only for their combination.

A different formalism of affordances was introduced in [Steedman, 2002]. Steedman's formalism presents a representation for objects in terms of their affordances using Linear Dynamic Event Calculus, which is a formalism for reasoning about causal relations over events.

The author builds the basis of his formalism on linear logic implication ($\multimap$), building on research by [Bibel et al., 1989] and others. For example, turning on and off a TV can be defined as:

$$\begin{aligned} on(x) &\multimap [button\text{-}press(y, x)]off(x) \\ off(x) &\multimap [button\text{-}press(y, x)]on(x) \end{aligned}$$

Linear implication representation models the update effects of actions. For example, if the initial situation was: $tv(t) \wedge off(t)$, then the action $button\text{-}press(person, t)$ will result in: $tv(t) \wedge on(t)$. It can be seen that this representation can be related to the STRIPS formalism introduced in [Fikes and Nilsson, 1971].

To represent affordances, Steedman first defines actions as functions derived from the previous events definitions. For example, $button\text{-}press(y, x)$ can be defined as:

$$button\text{-}press(y, x) \rightsquigarrow \left\{ \begin{array}{l} off(x) \multimap on(x) \\ on(x) \multimap off(x) \end{array} \right\},$$

where $\rightsquigarrow$ means "yields". Then he goes on to define affordances of object-concepts as the set of all such functions for the respective objects.

For example, assume beside the $button\text{-}press(y, x)$ action there is also a $channel\text{-}increment(y, x)$ action, then the affordance of TVs would be defined as:

$$Affordances(tv) = \left\{ \begin{array}{l} button\text{-}press \\ channel\text{-}increment \end{array} \right\},$$

The author finishes by defining, using Lambda calculus, the Gibsonian affordance-based object schemas as a function mapping the object (e.g., tv) into second-order functions from their affordances (e.g., $\textit{button-press}(y, x)$, $\textit{channel-increment}(y, x)$) to their results. For example, in our specific case, the tv-schema is: $\lambda x_{tv}.\lambda p_{Affordances(tv)}.px$ [Steedman, 2002].

This formalisation can also be suitable for planning using forward-chaining as mentioned in [Steedman, 2002].

Finally, another new important formalism was introduced in [Şahin et al., 2007] and [Cakmak et al., 2007]. In this research the authors first present the three possible perspectives through which to view affordances:

- Agent perspective: affordance relationships reside within the agent interacting in the environment through its behaviours
- Environment perspective: affordances over the environment as extended properties that can be perceivable by the agents
- Observer perspective: interaction of an agent with the environment is observed by a third party

The authors build their affordance formalism by first defining relation instances of the form: $(\textit{effect}, (\textit{entity}, \textit{behavior}))$. These reside within the interacting agent (i.e., all three components are assumed to be sensed by the agent). in [Şahin et al., 2007] For example, a blue book being grasped by a robot: $(\textit{grasped}, (\textit{blue-book}, \textit{grasp-with-right-hand}))$.

Since a single relation instance, which is normally discovered by one interaction with the environment, has very limited predictive ability for future interactions, the authors go on to propose four different types of equivalence classes, through which these relation instances can be bound together, thus enabling the discovery of affordances.

For example, the entity equivalence class is formed by those *entities* which support the generation of the same *effect* upon the application of a certain *behavior*. For example, both the blue book and a red one affording being grasped with the right hand:

$$(\textit{grasped}, (\left\{ \begin{array}{l} \textit{blue-book} \\ \textit{red-book} \end{array} \right\}, \textit{grasp-with-right-hand})).$$

The relation above can be represented as follows: $(\textit{graspeded}, (< \textit{*-book} >, \textit{grasp-with-right-hand}))$, with $< \textit{*-book} >$ denoting the derived invariants of the entity equivalence class [Şahin et al., 2007].

Similarly, the authors define behaviour equivalence. For example, if the *effect* is the same if the book is grasped with any hand, we have the relation: $(grasped, (< *\text{-}book >, grasp\text{-}with\text{-}*\text{-}hand))$.

Affordance equivalence is defined by a desired *effect* being accomplished through possibly different $(entity, behavior)$ relations. [Şahin et al., 2007] For example, the entering (of an enclosed space) affordance can be accomplished by either going through a gate or jumping over a fence:

$$(entered, \left\{ \begin{array}{l} (< gate >, < go\text{-}through >) \\ (< fence >, < jump\text{-}over >) \end{array} \right\}).$$

Finally, the fourth type of equivalence presented is effect equivalence.

The authors then finally go on to define affordances, from the agent's perspective, as follows [Şahin et al., 2007]:

" *An affordance is an acquired relation between a certain* $<$ *effect* $>$ *and an* $<$ $(entity, behavior)$ $>$ *tuple, such that when the agent applies an* $(entity, behavior)$ *within* $<$ $(entity, behavior)$ $>$*, an effect within* $<$ *effect* $>$ *is generated.*

The paper also presents how this operator can be used for planning. As opposed to STRIPS operators which are indexed by the action, the affordance operator in [Şahin et al., 2007] is indexed by its effect. For example, the same entering affordance would be represented as the following operator:

$$\begin{array}{l} (index : entered \\ \ effect : entered \\ \qquad\qquad\qquad (entity : gate, behavior : go\text{-}through) \\ \qquad\qquad\qquad (entity : fence, behavior : jump\text{-}over) \\ ) \end{array}$$

As opposed to this, the same setting needs to be represented by two different STRIPS operators, one for the action *go-through*, and one for the action *jump-over*, each indexed by its action [Şahin et al., 2007].

Sahin et al. claim that this affordance formalism has an advantage over STRIPS for planning where the environment is assumed to be perceived before the action, and where the plan is a sequence of actions.

Later work [Chemero and Turvey, 2007] presented a detailed comparison between the various competing affordance formalisms with the help of hyperset graphs. For more details on this comparison, please refer to [Chemero and Turvey, 2007].

3.1.2 Affordance Models in Applications

Despite the lack of agreement and variations on the definitions of affordances, over the years there have been numerous affordance-based approaches to the design of artificial agents. We will touch here on several different areas of research in robotics where affordance models have been used. This is not intended in any way as an exhaustive list, but more to present the many various tasks in which the concept of affordances can be employed and so to argue for their usefulness in robotics.

One area were affordances have been used was in the study of object avoidance and the traversal affordance. One example is the work in [Ugur et al., 2007]. In this study, a robot learned the traversability affordance of a room full of objects and used it in order to navigate around. The objects in the room included spheres, cylinders, and boxes, and the robot learned to avoid contact with non-traversable objects (e.g., boxes, upright cylinders), and to roll traversable objects out of the way (e.g., spheres). These affordances were learned from exploring an environment with a few objects and using features extracted from range images from 3D scans of the environment. SVMs are then used to classify features into the affordance categories (i.e., traversable or not) [Ugur et al., 2007]. The authors then expand on this research on the traversability affordance, including work on using it for multi-step planning in a room with obstacles in [Ugur et al., 2009] and while evaluating their affordance formalism in [Cakmak et al., 2007].

Another area investigated is the learning of object affordances in grasping and object manipulation setting. For example, the work in [Cos-Aguilera et al., 2005] presents a motivation-driven method for learning by modelling a motivational state and internal physiology. Then, by having the agent interact with objects, they measure the effect of the interactions on the motivational state. A similar approach is also presented in [Cos-Aguilera et al., 2003].

Grasp affordances were also studied in other works, including learning object-specific grasp affordance densities [Detry et al., 2009] and then extending the method to refine grasp affordance models by experience [Detry et al., 2010]. In these cases, the authors use grasp affordance as the relative object-gripper configurations that yield stable grasps, and represent them probabilistically with grasp densities, corresponding to continuous density functions defined on the space of 6D gripper poses. Grasps are sampled randomly from a density and are executed, and an importance-sampling algorithm learns a refined density from the outcomes of these experiences, where poor grasping solutions are downgraded [Detry et al., 2009].

There is work on affordance learning for visual perception systems, such as in

[Fritz et al., 2006]. Here the authors study the learning of causal relationships between visual cues, 3D features of visual entities, and predictable interactions, using both 3D and 2D information. For example, the fillable affordance can be determined for a coffee cup. Visual cues used for this purpose include colour, shape, 3D information, and SIFT (Scale Invariant Feature Transform) descriptors of the input image [Fritz et al., 2006].

Finally, worth mentioning as a field where affordances are used in robotics is that of tool use by robots. One of the earliest and more influential works include the one in [Stoytchev, 2005]. This work used a behaviour-based approach to ground the tool affordances in the behavioural repertoire of the robot. First, during an exploration stage, the robot chooses randomly different exploratory behaviours and applies them to the tools and observes the effects on the environment. Based on this a model is learnt that can then be used to solve tool-using tasks [Stoytchev, 2005]. Later work [Sinapov and Stoytchev, 2007] expands on this idea to have the robot learn the effects of its action with a tool and identify which frame of reference is useful for predicting these effects. The affordance model learnt is able to generalise and perform well even with tools that the robot has not experienced before [Sinapov and Stoytchev, 2007]. More research on tool use followed, examples including a Bayesian learning of tool affordances based on generalization of functional feature to estimate effects of unseen tools [Jain and Inamura, 2013], and a relational approach to tool-use learning in robots [Brown and Sammut, 2013], though this latter one making a weaker link to affordance models.

More references to research in affordances will be given in the following chapters where this will be related to the chapters.

3.2 Relational Learning and Models in Robotics

Although not that numerous, there has been research on relational learning and models in the field of robotics. We will present here some of the various tasks in which relational learning has made a contribution to solving various robotics tasks, and so argue for this approach in robotics. More specific examples of the state-of-the-art will be presented in the following chapters where this work is related to the one introduced there.

One of the most known works in relational robotics is the KnowRob [Tenorth and Beetz, 2009] system. It is a knowledge processing system particularly designed for performing everyday manipulation tasks in robotics. It consists of a first-order knowledge representation system based on description logics that provides mechanisms and tools for action-centered representation, for

automated acquisition of grounded concepts through observation and experience, for reasoning and managing uncertainty, and for fast inference. The KnowRob ontology, defined using the Web Ontology Language (OWL), is used to describe the robotics and household domains. Reasoning is performed with the help of the Prolog logical language. The system also allows for probabilistic reasoning by using the ProbCog[1] library, which implements statistical relational learning methods. The system is also able to add to its knowledge base by using Web sources, and has been deployed in several real life applications in a kitchen environment [Tenorth and Beetz, 2009].

Related to the above work, is that of equipping robot control programs with first-order probabilistic reasoning capabilities [Jain et al., 2009]. Here the authors present an architecture that provides a coupling between plan-based robot controllers and a probabilistic knowledge representation system based on statistical relational learning. The representation formalism the system uses for this purpose is Markov logic networks (MLNs). Thus, a robot can deal both with a high degree of complexity, by the use of first-order logic with models as general as possible, and be able to model uncertainty in the environment [Jain et al., 2009].

In the area of probabilistic planning there has been research on learning planning rules in noisy stochastic worlds [Pasula et al., 2004, Zettlemoyer et al., 2005], in a setting of a 3D simulated blocks world. The representation presented is an extension of STRIPS rules, with the addition of deictic references, consisting of variables and restrictions, which allow for better abstractions in the rule model, and the addition of a noisy outcome that implicitly models all rare and complex potential outcomes of rules. A learning algorithm that can create rules while also learning derived predicates is introduced and evaluated in a blocks world simulator [Pasula et al., 2004, Zettlemoyer et al., 2005]. Since the noisy indeterministic deictic rules introduced extend probabilistic STRIPS operators, they are related to our affordance formalism as we use it for planning, that also is based on probabilistic STRIPS. However, the approach in [Pasula et al., 2004, Zettlemoyer et al., 2005] is less suited than relational affordances for non-planning tasks, such as object search for example, since it does not define probability distributions over actions given a context (action rule preconditions).

Building up on the previous paper is research on integrated motor control, planning, grasping and high-level reasoning in a blocks world using probabilistic inference [Lang and Toussaint, 2010a, Toussaint et al., 2010]. This work integrates the previous probabilistic planning methods to a coherent control, trajectory optimization, and action planning architecture, using the principle

[1] http://ias.in.tum.de/research/probcog

of planning by inference across all levels of abstractions, and evaluating the approach in a real blocks-world setting with a Schunk robotic arm [Lang and Toussaint, 2010a, Toussaint et al., 2010].

The work on probabilistic planning mentioned above, as well as our work on relational affordances when used for planning tasks, falls in the area of partial model-based planning. A more extensive list of related work in the area of planning is presented in Section 6.7. We will mention here some additional related research on several different planning topics. Research on planning in nondeterministic domains under partial observability via symbolic model checking [Bertoli et al., 2001] presents an algorithm to generate conditional plans guaranteed to reach a goal with uncertainty in the initial conditions and action effects, and partial observability. This algorithm searches through an and-or graph induced by the domain, and is integrated in a planner based on binary decision diagrams and symbol model checking techniques. Research on online learning and exploiting relational models in reinforcement learning [Croonenborghs et al., 2007] proposes a representation for the transition and reward function that can be learned online, and then it can learn the world model, which is represented by the conditional probability distributions of the state variables by learning a decision tree. Research such as planning and acting in partially observable stochastic domains [Kaelbling et al., 1998] tackles the task of choosing the best actions in a partially observable stochastic domain, and proposes an algorithm for solving a partially observable MDP. To tackle problems involving sequential tasks one of the approaches used is dynamic Bayesian Networks. Research on relational dynamic Bayesian Networks [Sanghai et al., 2005] extends dynamic Bayesian Networks to first-order logic. This latter research also proposes two new forms of Rao-Blackwellised particle filtering which can be used for inference in relational dynamic Bayesian Networks. This planning research mentioned above is related to the area of probabilistic planning, and so related to our planning approach in Chapter 6 which relates to probabilistic-STRIPS, and thus some of these approaches could be used for a table-top planning setting as in Chapter 6. However, none of these build models that integrate the motor and perception capabilities of the robot as affordance models do. Also, they cannot tackle non-planning tasks, such as object search, which our more general affordance model formalism as presented in Section 3.4 can do.

Research on affordances is also related to research on object action complexes (OACs), which aims to integrate the low-level robotic control with high-level artificial intelligence planning. OACs aim to achieve this by pairing objects and actions in a single interface representation. Research on object action complexes as an interface for planning and robot control [Geib et al., 2006] represents OACs in linear dynamic event calculus by defining the high-level domain

actions, high-level domain properties, and the exogenous domain properties, which are provided by external (low-level) sources. The model then defines action preconditions and effects axioms in linear dynamic event calculus. The low-level control provides to the high-level system instantiated state transition fragments, which are small fragments of the planning domains state transition function. To learn the action representation, the system uses associative nets in order to learn from these instantiated state transition fragments an association between the linear dynamic event calculus actions and action preconditions axioms and the action effect axioms, allowing to learn the state transition model. Research on OACs: grounded abstractions of sensorimotor processes [Krüger et al., 2011] formalises the OAC definition, shows the relationship between the OAC model, the sensed, and the actual world, and defines an OAC as a triple of an identifier, a attribute space transition function, and a past statistical measure of success. This paper also defines several example OACs, including several pushing and grasping actions, and algorithms for learning such models. OACs have also been used in relation to work on solving the problem of object segmentation and the extraction of object shape [Kraft et al., 2008]. This is achieved by the active exploration of the robot where the robot actions are linked to the visual and haptic perception of objects through OACs. OACs are very related to affordances, in the fact that they too take into consideration the low-level robot control and so the capabilities of the robot. Since the main component of the OAC triple is the attribute space transition function, this is potentially less expressive than the joint probability distribution over object properties, actions, and effects, that we will use as affordance models as shown in Section 3.4.

Related to affordances and OACs, there is also research on learning the semantics of object-action relations by observation [Aksoy et al., 2011], which tackles the problem of recognising human manipulation and transferring this to a robot. The approach solves this problem by creating relational scene graphs to hold spatial relations between image segments, and a semantic event chain transition matrix stores changes to spatial relations during the temporal sequence of the human demonstration. It then shows of case study of learning the rules of an action sequence with no supervision, and then executing the actions.

Learning affordance models is also related to action model learning. Action model learning is concerned with modelling the preconditions and effects of actions from a training set of states and action observations. These models are generally represented in an action description language, such as STRIPS or PDDL among others, and they are then used in planning settings. There has been much research in the field of action model learning, and we will mention here some of this work. The work of [Shahaf et al., 2006] presents tractable exact algorithms for learning the preconditions and effects of actions in partially observable domains. These algorithms are applicable in deterministic

action domains, including STRIPS action models. An algorithm called ARMS [Yang et al., 2007] was developed for automatically discovering action models given a set of observed plans. This algorithm does not require partial intermediate states to be provided, building a statistical distribution of frequent sets of actions in the observed plans, which it uses to build and solve a weighted MAX-SAT problem.

Autonomous learning of action models for planning is also studied in [Mehta et al., 2011], where the authors present two frameworks for learning action models, a mistake-bounded planning framework and a planned exploration framework. In the former framework, the goal is to learn a model with at most a polynomial number of faulty plans given a planner, a simulator, and a planning problem generator. In the latter, a problem generator is not available, and the goal is to design planning problems and find solutions which converge with at most a polynomial number of planning attempts. The work of [Rodrigues et al., 2011] tackles relational action model learning and planning integration. It integrates an incremental action model learning with action selection in the context of agent adaptive behavior. The agent uses active exploration where actions are chosen to improve the action model, and the relational representations of the approach facilitate transfer learning. However, it has to be noted that our affordance learning approach has some differences compared to action model learning, which will be explained in more detail in Section 6.3. For example, as opposed to learning STRIPS rules, our affordance effects represent relative changes in one or more object properties due to an action, and the affordance model also allows us to model relations between different object properties irrespective of the action, and similarly relations between certain effects.

In the robotics area of object recognition, there was recent work on object category recognition by a humanoid robot using behaviour-grounded relational learning [Sinapov and Stoytchev, 2011]. In this research, the robot explores through its set of exploratory behaviours a set of household objects, and records the proprioceptive and auditory sensory feedback produced by the interactions. This data is used to estimate multiple measures of similarity between the objects. The classification of the object in the different household categories is done with the help of a graph-based recognition model is trained by extracting relational features from the estimated similarity relations. The system was evaluated successfully with an upper-torso humanoid robot and a large number of household objects [Sinapov and Stoytchev, 2011].

In the area of object manipulation, an example of relational learning to solve a specific robotic task is the paper on learning to manipulate articulated objects in unstructured environments using a grounded relational representation [Katz et al., 2009]. Here the robot interacts with the articulated objects in the

environment in order to acquire information about their kinematic structure. This is represented with a relational representation, where predicates represent the different types of possible joints. To learn how to manipulate the objects, reinforcement learning is used with the help of relational Markov decision processes (RMDPs). A policy is acquired incrementally by choosing the best action to perform for the robot, and receiving a reward for each degree of freedom discovered. The system was tested with real-world articulated objects acted upon by a single arm manipulator [Katz et al., 2009].

In the area of navigation, one example of relational learning is the work on learning relational navigation policies in [Cocora et al., 2006]. The paper proposes the learning of relational decision trees from example paths as abstract navigation strategies. This approach is able to generalise navigation plans, as opposed to path planning given just the initial setting and goal of the robot, and to be transferred to other environments. Example navigation plans can be computed with the help of RMDPs, and relational learning techniques are used to obtain the relational decision tree representing the abstract navigation policy. The approach was tested with a real robot in an office building environment [Cocora et al., 2006].

As we have seen, relational learning and modelling has been used for many of the different areas of robotics to help achieve various different tasks. More references to research in relational robotics will be given in the following chapters where this constitutes work related to the one presented in the respective chapters.

3.3 Context of Our Work

With all the different interpretations and uses of affordances which we just introduced, our affordance setting will be an extension of three different works [Lopes et al., 2007, Montesano et al., 2008, Krunic et al., 2009]. In these papers, affordances model the relations between object properties, actions and effects for single objects with the aid of Bayesian Networks in order to achieve several manipulation tasks.

In [Lopes et al., 2007] affordances are used to build an imitation learning algorithm. The robot first learns a task independent affordance model by interacting with the world, which it models as a Bayesian Network. The robot can then observe a demonstration of an action by another agent (a human). In order for the robot to imitate this action, it needs to interpret it in terms of its own action repertoire, relying on the observed state transition or corresponding effects. The robot uses this methodology in order to tackle a simple imitation

game, where the robot needs to perform one of its actions on objects placed on a table depending on the type of the object [Lopes et al., 2007].

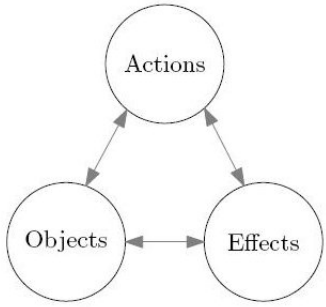
The research on learning object affordances: from sensory–motor coordination to imitation [Montesano et al., 2008] expands on the previous setting. The work presents a general model for learning object affordances using Bayesian networks integrated within a general developmental architecture for social robots. This approach is able to deal with uncertainty, redundancy, and irrelevant information. The approach again is tested with an imitation game where the robot needs to obtain the same observed effects on objects as a human demonstrates [Montesano et al., 2008].

Finally, the work in [Krunic et al., 2009] extends an affordance network to incorporate words, thus associating meaning to its manipulation tasks. The model uses verbal descriptions of a task and temporal co-occurrence to create links between speech utterances and the involved objects, actions and effects in the affordance model. The robot is able to form useful word-to-meaning associations, even without considering grammatical structure and in the presence of recognition errors. The words can then be used to instruct the robot to perform a task [Krunic et al., 2009].

In these works, affordances are modelled as relations between the following three variables: the set of *objects* and their *properties* as being detected by the robot sensors: $O = \{o_1, o_2..., o_n\}$, the repertoire of *actions* available to the robot, $A = \{a_1, a_2..., a_n\}$, and the *effects* of performing those actions $E = \{e_1, e_2..., e_n\}$ as detected by the sensors as changes in object features. Using an affordance model, given two of these variables, one can predict the third. Thus, as mentioned in [Montesano et al., 2008] , affordances allow to perform three tasks:

- predict the outcome (and plan) an action (infer E, given O and A)
- recognize a performed action (A, given E and O)
- select objects according to a task requirement (determine O, based on observed A and E)

For example, affordances can be used for imitation learning [Lopes et al., 2007] or action prediction by computing the *maximum a posteriori probability* (MAP) estimate $\arg\max_A P(A|O,E) = \arg\max_A \frac{P(A,O,E)}{P(O,E)}$, given the values of O and observing the E. A generic affordance model and its three uses is shown in Figure 3.1.



Inputs	Outputs	Function
(O, A)	E	Effect prediction
(O, E)	A	Action recognition/planning
(A, E)	O	Object recognition/selection

Figure 3.1: Affordances: relations between objects (properties), actions, effects [Lopes et al., 2007, Montesano et al., 2008].

The object affordances are usually learnt by a robot through an initial exploration phase, during which the robot manipulates objects in the environment using its set of actions, and perceives the effects of its actions.

This exploration phase is called motor babbling. Its use in robotics was inspired from developmental psychology studies on child development, which investigated body babbling, the movement practice of infants gained through self-generated activity [Meltzoff and Moore, 1997]. In robotics, it refers to the robot generating random joint movements, and using these together with information from its vision system, to collect learning samples of its motor actions on the environment. This phase is followed by a learning phase, in which the robot uses the collected samples in order to learn a model of its actions on the environment [Demiris and Dearden, 2005, Saegusa et al., 2009].

Once learnt, the affordance models can be used to define the relationships between the robot and the environment through the robot's available sensing and motor capabilities [Lopes et al., 2007, Montesano et al., 2008].

3.4 Relational Affordances

We will now introduce our relational affordance concept. It is an extension of the affordance model of Figure 3.1, but, rather than using a propositional representation for object properties, actions, and effects, we now use a relational one. Thus a relational affordance is a joint distribution over a relational representation for O, A, and E. To define this joint probability distribution $P(O, A, E)$ we will use a relational representation in the form of a PPL program, which we will illustrate with the help of DCs-style syntax. We chose this affordance formalism because it can easily model a wide variety of settings, including the table-top manipulation and object search settings investigated in this thesis, because it can easily be extended to model the (spatial) relations between the objects in the scene, and finally because it can easily be modelled with the help of a PPL.

We will use $\mathcal{Z}$ to represent the set of all objects in the environment the affordances model. An uppercase $\mathtt{Z}$, optionally followed by a subscript, denotes a variable in the domain $\mathcal{Z}$, and a lowercase $\mathtt{z}$, optionally followed by a subscript, denotes a constant in the domain $\mathcal{Z}$.

The elements of an affordance model (object properties, action, effects) can be defined as follows. *Object properties* O are the set of all random variable atoms of the form $\mathtt{op_i}(\mathtt{Z_1}, ..., \mathtt{Z_{m_i}})$. The value of a grounded object property is computed from features extracted from the perception data of the robot before the action is executed: e.g., $\mathtt{distance}(\mathtt{ball}, \mathtt{book}) = \mathtt{5}$. Therefore, the computation of object properties takes into account the perception capabilities of the robot. The affordances also model thus how the robot perceives its world through its sensors.

The set of *actions* on object Z is denoted by: $A(Z)$. One action applied to object $\mathtt{Z}$ is denoted by the atom $\mathtt{a_i}(\mathtt{Z})$, where $\mathtt{a_i}(\mathtt{Z}) \in A(Z)$. The set of actions is determined by the motor capabilities of the robot.

Effects E are the set of all random variable atoms $\mathtt{e_i}(\mathtt{Z_1}, ..., \mathtt{Z_{n_i}})$. The value of a grounded effect is computed from features extracted from the perception data of the robot after the action is executed: e.g., $\mathtt{bounce}(\mathtt{ball}) = \mathtt{true}$. By computing the effects from perception data after the action execution, the effects take into account both the motor and the perception capabilities of the robot. Therefore, in line with Gibson's affordance definition, affordances model what the robot can do with the objects in its world.

If, for all $\mathtt{op_i} \in O$ and $\mathtt{e_i} \in E$, $\mathtt{m_i} = 1$, and $\mathtt{n_i} = 1$, we have a propositional affordance, which is similar to the one presented in Section 3.3. Otherwise, if there exists at least one $\mathtt{m_i} > 1$ or $\mathtt{n_i} > 1$ then we have a relational affordance.

This means that in a relational affordance, there exists at least either an object property atom or an effect atom with arity higher than one. In this thesis, we only consider single object properties and effects, and relational object properties and effects between pairs of objects. So, the arity of any of the $\texttt{op}_\texttt{i}$ and $\texttt{e}_\texttt{i}$ atoms will be either one or two, but the approach can be generalised.

Example 3.1. *Consider the following example where we have a table-top setting with three objects:* $\mathcal{Z} = \{\texttt{ball}, \texttt{book}, \texttt{cup}\}$. *Consider the object properties* `color`, `shape` *and* `distance`, *and the effects* `bounce` *and* `rel_displacement`.

In this setting, we can have, the following affordance model elements:

Object properties:

$\texttt{color}(\texttt{Z}) \sim \texttt{uniform}([\texttt{red}, \texttt{blue}]) \leftarrow \texttt{true}.$
$\texttt{shape}(\texttt{Z}) \sim \texttt{uniform}([\texttt{sphere}, \texttt{cube}]) \leftarrow \texttt{true}.$
$\texttt{distance}(\texttt{Z}_1, \texttt{Z}_2) \sim \texttt{gaussian}(4, 2) \leftarrow \texttt{true}.$

e.g.: $\texttt{color}(\texttt{ball}) = \texttt{red}, \texttt{shape}(\texttt{ball}) = \texttt{sphere}, \texttt{distance}(\texttt{ball}, \texttt{book}) = 5$

Actions: $\texttt{a}(\texttt{Z}) \sim \texttt{uniform}([\texttt{tap}(\texttt{Z}), \texttt{push}(\texttt{Z})]) \leftarrow \texttt{true}.$ *e.g.:* $\texttt{push}(\texttt{ball})$

Effects:

$\texttt{bounce}(\texttt{Z}) \sim \texttt{uniform}([\texttt{true}, \texttt{false}]) \leftarrow \texttt{true}.$
$\texttt{rel\_displacement}(\texttt{Z}_1, \texttt{Z}_2) \sim \texttt{gaussian}(2, 1) \leftarrow \texttt{true}.$

e.g.: $\texttt{bounce}(\texttt{ball}) = \texttt{true}, \texttt{rel\_displacement}(\texttt{ball}, \texttt{book}) = 2$

Relational affordances are defined by the joint probability distribution over O, A, E: $P(O, A, E)$. This joint probability distribution is normally learnt in the babbling stage, and it will be modelled by a PPL program. Then, we can ask for the probability of a given query.

Example 3.2. *Consider the relational affordance model defined by the joint probability distribution and DC program from Example 3.1. We can ask, for example, for the following joint probability distribution query:*

$$\begin{aligned} P(&\texttt{color}(\texttt{ball}) = \texttt{red}, \texttt{color}(\texttt{book}) = \texttt{blue}, \texttt{shape}(\texttt{ball}) = \texttt{sphere}, \\ &\texttt{shape}(\texttt{book}) = \texttt{cube}, \texttt{distance}(\texttt{ball}, \texttt{book}) > 3, \\ &\texttt{a}(\texttt{ball}) = \texttt{tap}(\texttt{ball}), \\ &\texttt{bounce}(\texttt{ball}) = \texttt{true}, \texttt{bounce}(\texttt{book}) = \texttt{false}, \\ &\texttt{rel\_displacement}(\texttt{ball}, \texttt{book}) > 0) = 0.0058 \end{aligned}$$

Using these relational affordance models allows us to answer complex queries. For example, assume in Example 3.2 that *bounce* is instead modelled by a Gaussian distribution and it represents the height the object bounces to. We can have a more complex task that needs to answer the question: What is the probability that I have a sphere given that it bounces higher for an (unspecified)

action than it would be displaced by a different action? And for which action pairs is the probability of a higher bounce greater than 50%?

Defining a PPL program, for example using DCs, will allow us to define the joint probability distribution $P(O, A, E)$ that defines the relational affordance model. We want to build this probabilistic program representing the joint probability distribution from the babbling data collected by the robot. Beside ProbLog and DCs, there are many other PPLs available including for example BLOG [Milch et al., 2005], Church [Goodman et al., 2008], IBAL [Pfeffer, 2001], and PRISM [Sato and Kameya, 1997], among others. However, none of these provide publicly available code that can learn the structure of a PPL from data. This problem has been tackled in the graphical models community. There are algorithms for learning graphical models, for example BNs, from data. However, determining the BN structure that optimally represents a given set of training data is an NP-complete problem [Chickering, 1996]. To tackle the problem of BN structure learning, there are two main approaches, search-and-score and constraint-based. The search-and-score approach uses a score function and examines different BN structures, selecting the one(s) with the highest score. Score functions are designed to give higher scores to structures best fitting the training data distribution, and this approach generally uses Markov Chain Monte Carlo algorithms to search through the model space. The constraint-based approach test triplets of the form $(A, B, \mathcal{S})$, with variables A and B, and subsets of variables $\mathcal{S}$ to check if A and B are independent given $\mathcal{S}$. These tests then define constraints on the BN structure, and this approach then selects the structure(s) satisfying the most constraints [Koski and Noble, 2012]. Publicly available implementation are provided by several software packages, including for example the BNT toolbox [Murphy et al., 2001] for Matlab, which provides the exhaustive search, K2, hill-climbing, MCMC structure learning algorithms among others. Therefore, to learn our PPL program from data, we will first use one of these structure learning implementations for BNs, followed by parameter learning as described in Section 2.1.2, after which we will model this learnt BN in a PPL. For a full review of many of the BN structure learning algorithms, please refer to [Koski and Noble, 2012].

Therefore, in order to create this PPL program, we will use the following steps:

- learn a (LCG) BN that represents the structure of the babbling data for the discrete or continuous domain
- write a PPL program that represents this (LCG) BN, where to generalise over the number of objects we introduce variables for objects
- add logical rules that reflect background knowledge

The relational affordance models integrate together many aspects of modelling the world the robot has to act in. Firstly, the robot capabilities are taken into account since the object properties, such as *shape* or *distance* from Example 3.1, are computed from data acquired by the robot sensors. The motor skills of the robot are taken into account in the model, since the effects, such as *bounce* from Example 3.1, are computed after the motor action on the objects. So what the robot can do, and what opportunities the environment presents to the robot, is part of the model. Secondly, the affordance model integrates both discrete and continuous data, which is necessary for a realistic and more general modelling of the environment in which the robot is placed. We have seen in Example 3.1, that the affordance model contains discrete object properties, such as *shape* which can take one of two possible values, and continuous effects, such as the relative distance between two objects. The modelling of the relationships between discrete and continuous variables is necessary in robotics settings, and we will show starting with Chapter 5 how this is achieved in our PPL models of relational affordances in various robotics settings. Thirdly, the relational affordance model adds the symbolic aspects of the world representation. With the help of a PPL, we model relations between objects, and by the use of variables, we are able to generalise to settings with various objects, and various number of objects. We will show how this is achieved with PPL code examples starting in the next chapter. All these three aspects constitute and are modelled with our relational affordances.

All the following chapters will incorporate the basic steps introduced above when building a PPL model representing the relational affordance model. Furthermore, depending on the tackled task, additional steps will be needed, as mentioned in the respective chapters.

The relational affordance models defining joint probability distributions over O, A, E will be used in Chapter 4, for action prediction in a single action setting where the temporal aspect does not need to be modelled, and in Chapter 7 for occluded object search, where the effects will not be specifically modelled. They are also used in Chapter 5 for modelling of simultaneous two-arm actions. The modelling of sequential two-arm actions in Chapter 5, and the planning setting of Chapter 6, will also require the definition of a state description and action representation based on these relational affordance models, as it will be shown in more detail in those chapters.

3.5 Conclusions

We have first introduced in this chapter the concept of affordances as introduced by Gibson, followed by the presentation of some of the main formalisms used to define affordances that have been introduced since. We have continued with showing how affordance model have been used for solving various robotics tasks. Secondly, we have also briefly introduced several works that used relational learning in robotics, in order to highlight their usefulness in solving robotics tasks. Finally, we focused on the affordance setting we will extend to a relational domain in the following chapters.

Chapter 4

Learning Relational Affordance Models

This chapter[1] introduces the core concept of relational affordances, and presents how such models can be learnt using SRL methods from data collected by the robot through a babbling stage. We then show how these relational affordance models can be used for simple object manipulation tasks, such as action prediction. We will first illustrate the learning of relational affordances in a discrete setting, and then evaluate our model in a table-top action prediction setting.

4.1 Introduction

Robotics aims to develop mobile, *physical* agents capable of reasoning, learning and manipulating their environment. Early approaches such as the well-known Shakey robot used the logical STRIPS representation [Fikes and Nilsson, 1971], and several other such symbolic representations have been used [Hertzberg and Chatila, 2008, Stulp and Beetz, 2008]. Approaches based on logic are effective at dealing with higher level knowledge needed for planning and reasoning, but the physical aspect of robots requires dealing with various kinds of *uncertainty*, typically not handled by such formalisms. These aspects include interpreting noisy sensors, processing

[1]This chapter is based on work previously published in [Moldovan et al., 2012a] and [Moldovan et al., 2012b]

image streams from cameras, controlling noisy physical actuators for manipulation, and in general, solving many *grounding* and *anchoring* problems [Coradeschi and Saffiotti, 2003]. The use of *probabilistic reasoning and learning* techniques is now widespread in robotics [Thrun et al., 2005], yet mostly without employing structured, logical representations. For this we need *statistical relational learning* (SRL) [De Raedt and Kersting, 2008, De Raedt, 2008] which combines logical representations, probabilistic reasoning and machine learning. Recent works have explored the use of SRL and shown how to effectively combine probabilistic and logical methods in robotics domains (e.g., a kitchen scenario [Jain et al., 2009]).

A promising approach for the skills development of humanoid robots is the learning of *object affordances*. Affordances are modeled as relations between three variables, *objects* (O), *actions* (A) and *effects* (E), such that given two, one can predict the third. Thus, generally, affordances can allow us to perform one of the following three tasks: predicting the *outcome* (and plan) an action, recognising a performed action, or selecting objects according to a task requirement [Montesano et al., 2008]. The introduction of affordances to robotics follows the developmental framework, which proposes to acquire new skills on top of previous ones by experimentation and interaction with the environment [Lungarella et al., 2003]. This allows the robot to perform more complex tasks and learn by imitation in a goal oriented manner [Montesano et al., 2008]. The typical scenario of affordance learning considers isolated objects in the environment, which is a reasonable assumption for many robotic tasks. However, there are several cases where it is necessary to consider the (spatial) relations between several objects such as: shelf sorting, placing groceries in a shopping cart and packing items in a bag. In this chapter we address such settings, which we refer to as *relational affordance learning* in multiple-object settings.

4.1.1 Problem Statement and Approach

This chapter introduces the concept of relational affordances, and describes how such models can be learnt and used in simple table-top object manipulation settings. For this purpose, we study the extension of the affordances model to include more than one object, where the robot's actions may affect several objects in the environment. Previous approaches adopt Bayesian networks (BNs) in order to encode the relations between an object's properties, actions and effects in a probabilistic manner [Lopes et al., 2007, Montesano et al., 2008, Krunic et al., 2009]. A straightforward extension to that model to handle multiple objects is to add more variables to the BN and perform structure learning and inference as in previous works. However, two issues must be

considered if this straightforward extension would be chosen: (i) a separate BN is learnt and instantiated for every distinct number of objects in the environment and (ii) the number of samples needed to learn a BN with more than three objects is very large. We address these issues using SRL (and probabilistic programming languages) [De Raedt et al., 2007], which allows to build just one model (probabilistic program) that supports inference for any number of objects, so the structure learning of several BNs is not needed and performing inference does not need to switch between BNs. The probabilistic program will define the joint probability distribution over O, A, E, which defines the relational affordance model. The SRL generalization also enables inference on scenarios with (spatial) relations between more than two objects, using only samples of scenarios with one and two objects.

In order to create and use a relational affordance model for multiple interacting objects in a table-top scenario, we will solve three different tasks (two learning tasks and an inference task):

- Task (i) is learning a single object affordance model representing the joint probability distribution $P(O, A, E)$ and modelling it with a PPL: *given:* a) a set of corresponding O, A, E values collected from exploratory action executions in single object environments, *find:* b) an affordance model of single objects, modelled in a PPL
- Task (ii) is learning a multiple-object relational affordance model: *given:* a) the single object affordance model learnt in Task (i), b) a set of corresponding O, A, E values collected from exploratory action executions in two-object environments, and c) background knowledge representing a set of constraints on the actions (e.g., grasp not allowed for objects that are very close to each other), *find:* d) a relational affordance model of multiple-object settings, modelled in a PPL
- Task (iii) is the action prediction inference task used to evaluate our model: *given:* a) the relational affordance model learnt in Task (ii), b) an initial scene from which the set of object properties values O can be extracted, and c) a target goal, given as E, *find:* d) predict the action of the robot, given by: $\arg\max_A P(A|O, E)$.

To solve these tasks, several steps need to be performed, as shown in Figure 4.1. These are:

- 1a) learn a Bayesian Network (BN) from single-object exploratory data,
- 1b) from the single object BN model, build a single object affordance model in a PPL,

- 2a) learn a BN from two-object exploratory data,
- 2b) from the two-object BN model, together with the use of background rules and the use of the single object model for cases with no interactions, build a multiple-object relational affordance model in a PPL,
- 3) perform action prediction using the relational affordance model.

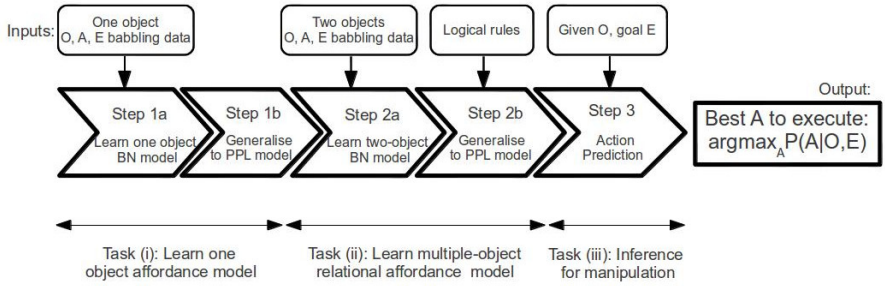


Figure 4.1: Pipeline for learning relational affordance models.

All of these steps will be explained in detail in this chapter. Once we have created this relational affordance model (which can be found in Appendix A), we will evaluate it in a multiple object, action prediction setting.

In this chapter, for illustrating these concepts we use the simulator of the iCub robot [Metta et al., 2008] to perform experiments, where the humanoid robot is capable of discovering the affordances associated with manipulation actions (*grasp*, *push* and *tap*), applied to objects with different properties (*color* and *shape*). The *effects* include *displacement* and *orientation changes* for each object and *contact* between objects. These effects are closely related to the multiple object scenarios mentioned above, where the position and contact information between objects guide the robot to select the appropriate object and action for the subsequent step. Other applications of relational affordances that we will present in later chapters will also make use of the PR2 simulator, as well as of the iCub robot in a real setting.

An example of the type of applications for multi-object affordance models is the *shopping shelf scenario*, depicted in Figure 4.2. One example of a possible task for the robot is to place the magenta object onto a shelf at the given target location. This involves first pushing the other objects to the left to make space (1), followed by pushing the magenta object in its target location (2). This type of more complex applications involving planning for sequences of actions will be explored more in depth in Chapter 6.

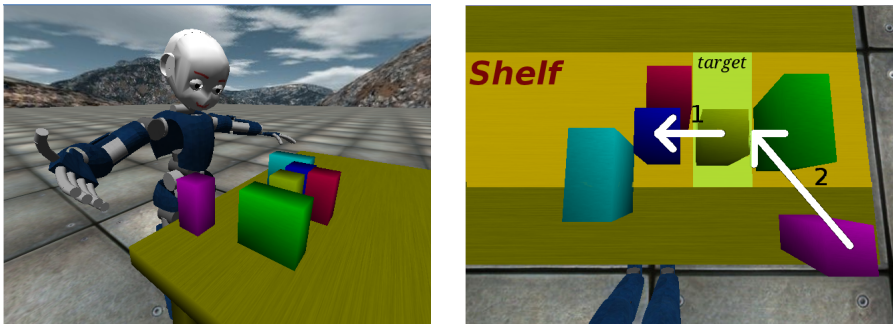


Figure 4.2: Shelf scenario with action sequence for object placement.

4.1.2 Contributions and Outline

There are several contributions brought by the introduction of relational affordance models. Affordance models using BNs need to be tailored to the specific number of objects in a scene, and their size increases with this number. The contribution of relational affordance models is the use of SRL methods for extending affordance models to multi-object scenes by robustly going from two-object interactions to a variable number of objects by generalization over objects and the modeling of probabilistic aspects. SRL provides ease of modeling and increased comprehensibility by allowing the transfer of the model structure to other domain sizes. The experiments using an iCub robot presented later in this chapter will show in multi-object scenarios that this approach gives comparable results to a tailored BN, while maintaining the flexibility of not needing retraining and the robustness for large numbers of objects.

The rest of this chapter is organized as follows: Section 4.2 presents the technical setting and basic skills of the robot used for illustrating and evaluating the relational affordance concept and Section 4.3 shows single object affordance modelling. Section 4.4 includes the main contribution of this chapter: it explains the extension toward relational, multi-object affordance models and describes examples of two-object interactions. Section 4.5 presents experimental results in a multiple-object setting. Finally, Section 4.6 presents related work and we conclude the chapter in Section 4.7.

4.2 Basic Skills of the Robot

We employ, both in a real setting and in simulation, the *iCub humanoid robot*, which has a head with two cameras, two arms and two legs. We use only one arm, the cameras and the software modules that provide: (i) motion control to reach a target position [Pattacini et al., 2010], (ii) image segmentation [Christoudias et al., 2002] and (iii) stereo triangulation. We build on these elements the basic skills of the robot: motor skills to perform the actions and perceptual skills to measure object features and effects.

We assume the robot is provided with a set of core motor actions with parameters adjusted after self-experience [Montesano et al., 2008]. The motion of the hand to a target position is provided by visual skills. The hand is moved to the target position by a minimum-jerk Cartesian controller which reaches a position as close as possible to a given rest position while coping with the kinematic constraints of the robot, such as joint limits, damage avoidance and hand orientation [Pattacini et al., 2010]. The action is performed after the Cartesian controller has terminated. The action execution is preprogrammed due to the limitations of the simulator and the complexity of the iCub's hand. The distance over which the hand is moved during an action was chosen to be the same as the size of the smallest object on the axis over which the movement is done. For the *push* and *tap* actions, this is $4cm$, while a *grasp* action moves the object diagonally by a comparable value.

The perceptual skills of the robot include color segmentation and 3D object localisation. The color segmentation algorithm relies on a synergistic approach that combines a confidence-based edge detector and mean shift segmentation [Christoudias et al., 2002]. We apply the image segmentation algorithm on both cameras in order to find the enclosing region of objects on each image. Then, the centroid of the segmented region is extracted on both cameras in order to perform stereo triangulation that provides the 3D position of the object's centroid, which is the target position to execute the action.

The overall architecture of the system can be seen in Figure 4.3. The perception module retrieves the two eye camera images from the iCub simulator. It runs the color segmentation and 3D object localisation algorithms in order to obtain a set of object IDs, object sizes and locations from the stereo camera images. The decision module uses the data from the perception module and the required effects given by the user in order to first compute the values for the object properties O, and effects E, and then does inference using the PPL relational affordance model in order to compute the most likely action to execute: $\arg\max_A P(A|O,E)$. This obtained action is passed on to the motion control module which executes the hand motion using the Cartesian controller.

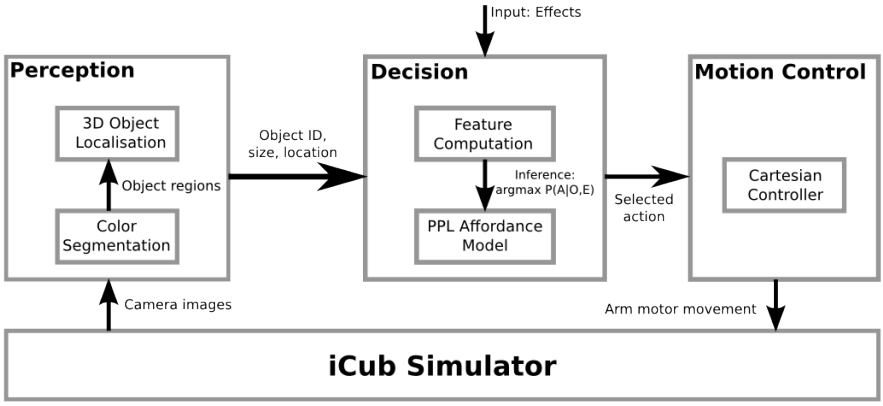


Figure 4.3: Overall architecture schema of the system.

The real setting, with the iCub manipulating objects in over 100 scenarios, was used as a proof-of-concept demonstrating that our approach can be used as well in a physical setting with just minor adjustments. However, because of practical constraints and for easiness of illustration, most of the results reported in this chapter are obtained in simulation.

4.3 Affordance-based Models

We now describe the variables of our affordance model, illustrated in Figure 4.4 with the objects' position before (l) and after (r) an action (tap) execution.

Figure 4.4: Relational O before (l), and E after the action execution (r).

The set of object properties O consists of the following: *shape*, and (the relational properties) the relative distance ($initRelDist$) and orientation ($initRelOri$) between each pair of objects. After an initial domain analysis, the numerical

values for $initRelDist$ and $initRelOri$ were discretized as follows: the relative distance in 4 clusters separated at $6cm$, $10cm$ and $14cm$, and the orientation angle in 8 clusters of $45°$. The object property $initRelDist$ is measured centre-to-centre as shown in Figure 4.4(l) in white. Given that our object sizes on the two table-plane axes are either $4cm$ or $8cm$, the four clusters are chosen so they can differentiate overlap of the centre of the objects on one of the axis given the object types involved. For example, for the cluster with values between $0cm$ and $6cm$ there's definite overlap on one axis for any two objects, while for the cluster between $10cm$ and $14cm$ there is overlap only if one of the objects is the bigger object. Note that since we use the perception of the robot to compute the centroid of an object, uncertainty is introduced, and in the case of complex shapes there can be bigger errors when computing centre-to-centre distances between objects. The 8 possible values of $initRelOri$ can be seen in Figure 4.4(l) in yellow. In this particular case, the value of $initRelOri$ is: WNW. We use two object shapes, cubes and rectangular prisms, denoted by their 2D shape as observed by an observer placed above the table (i.e., *square* and *rect*, respectively).

We consider three basic core motor actions: $A = \{push, tap, grasp\}$: *tap* (right-to-left hand movement), *push* (away movement) and *grasp* (pick-up, move to right and away, then release). Each moves the iCub hand over a preprogrammed orientation and distance, with parameters adjusted after self-experience, but as shown in later chapters the actions can also be easily parameterized.

The effects E are measured as differences in object attributes before and after the action is performed. We measure: (i) $dispMag$: the magnitude of the object displacement relative to its initial position, (ii) $dispOri$: the orientation angle of the displacement, and (iii) *contact*: the contact between objects, computed from the intersection between the segmented regions in both right-eye and left-eye iCub images. The displacements are computed for the centre of each object. The displacement and its angle orientation are also clustered, with the displacement clusters separated at $1cm$, $3cm$ and $5cm$, and the orientation angle in 8 clusters of $45°$. Given our object sizes of either $4cm$ or $8cm$ on the two table-plane axes, the four displacement clusters are meant to represent in order: negligible displacement, displacement comparable to half the length of the smallest object, displacement comparable to the length of the smallest object, and displacement greater than the length of the smallest object. These are shown in Figure 4.4(r), which overlays the initial objects' positions over their final positions.

As the robot itself is not mobile and the arm has a specific action range, each $a_i \in A$ can be performed when the object is located in a specific action space. This is learnt by exploratory manipulation of objects in various locations, and measuring and k-means clustering [Lloyd, 1982] (using Matlab) of the effects

(e.g., displacement) in three clusters. Figure 4.5 shows the action space for a *tap* action. It presents the three clusters of the effect values for objects at various locations on the table during the babbling phase. The x and y-axis coordinates are in meters.

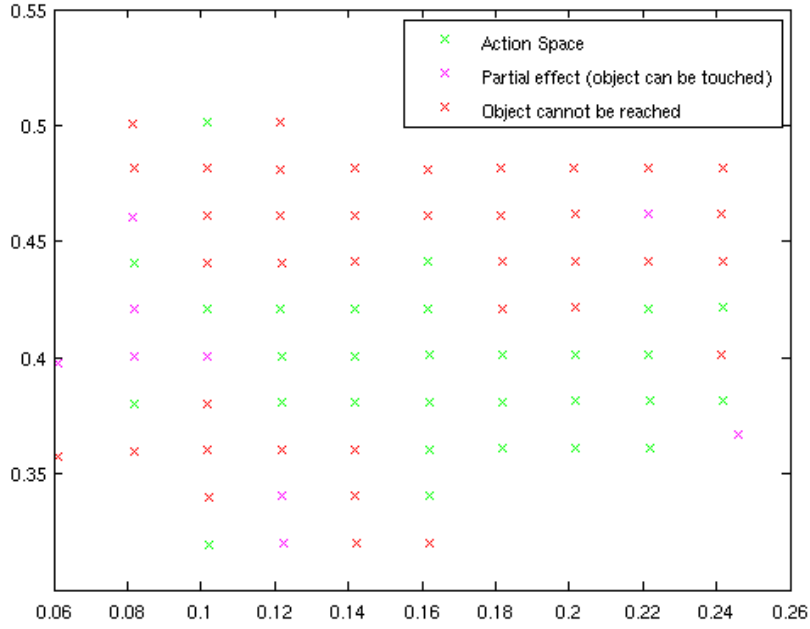


Figure 4.5: Action space for a *tap* action

The lowest displacement values correspond to the object not being reached, and the medium ones to where the action cannot be performed fully. The cluster with the highest values (i.e. effects are consistent) determines the action space. This action space can also be enforced by the introduction of (logical) rules. For example, for a *tap* action the x-coordinate of the (centre of) the manipulated object needs to be between $12cm$ and $22cm$, which can be expressed by: $12 \leqslant coordX(Obj), coordX(Obj) \leqslant 22$.

4.3.1 Learning Affordances for One Object

We will start introducing here the first part of our multiple step process for creating a relational affordance model, namely **Step 1a**, learning a BN

representing single object affordances.

We will continue showing in a later section how this single object affordance can be modelled with the aid of SRL, which constitutes Step 1b of our process, in order to build a model that can be used for multiple objects when there are no interactions. The reason a BN is learnt first, is to create the structure of the PPL program, as currently there is no way to learn a PPL program directly from a collected dataset of observations. The BN can then be converted into a ProbLog program, using for example the method we previously described in Section 2.3.1.

To learn an affordance model, the robot will start by exploring single object environments. Through a behavioural babbling stage, the robot executes one of its actions A on one object placed in front of the robot at various positions. The robot collects the values of the object properties O for the respective object, and evaluates the effects of its action E. Through this method, it collects a set of O, A, E values for its chosen exploratory action. The robot executed more than 100 such exploratory actions, obtaining sets of O, A, E values. One such example set, for a tap action, is shown in Table 4.1.

Table 4.1: Example collected O, A, E data for a tap action with one object

O. Properties	**Action**	Effects (raw values)	**Effects (discretised)**
$shape : square$	tap	$dispMag : 4.27cm$ $dispOri : 177°$	$dispMag : 2$ $dispOri : WNW$

To choose the settings for our babbling stage, we placed the single object in front of the robot in and around its action space. For each setting, we varied the x and y-coordinates of the object by $2cm$. This value roughly corresponds to the tolerance of the robot's inverse kinematics trajectory planning to reach a goal position, as well as to the error values in the 3D localisation of the object due to its perception. The over 70 different settings for a *tap* action can be seen in Figure 4.5. Note that to increase the accuracy of the model we could repeat these settings several times to collect more data, however this requires that the robot spends more time in this babbling stage.

During this behaviour babbling phase the robot also learns the action space for each $a_i \in A$. Only the subset of the data in the action space will be used to learn the affordance model to represent the environment.

For learning the affordance model, we use a BN [Pearl, 1988] to encode the dependencies in the affordance model, using a two-step approach similar to [Montesano et al., 2008, Lopes et al., 2007]. The first step consists of defining

the structure of the BN. In most cases this comes from domain knowledge of the problem or task to be modelled. Alternatively, one can also learn the structure of the BN from the dataset collected through behavioural babbling. In a first phase we induced the connection structure of the BN, using the K2 algorithm (using Matlab), which is computationally faster than the alternative MCMC approach [Montesano et al., 2008]. K2's greedy nature may generate an incomplete structure which can easily be corrected using domain knowledge.

The K2 algorithm was chosen since our experimental evaluation compares our relational affordances approach to the BN modelling such as in [Montesano et al., 2008, Krunic et al., 2009]. Since [Montesano et al., 2008, Krunic et al., 2009] use the K2 algorithm for structure learning, we want to use a similar approach such that any difference in the results will not be due to a different structure being learnt, but to our relational approach. Note that as future work, several other structure learning algorithms such as hill climbing or MCMC to search the space of the directed acyclic graphs, or structural EM, can be investigated.

The BN structure of the our single object affordance model is depicted in Figure 4.6.

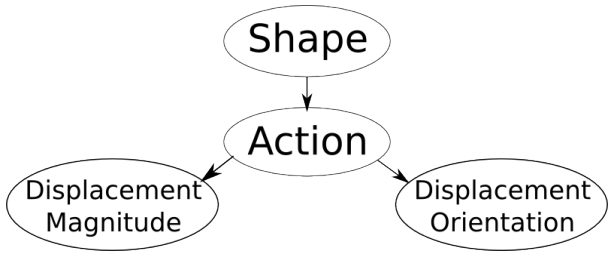


Figure 4.6: BN representing affordances for the one-object setting.

The second step, similar to [Montesano et al., 2008, Lopes et al., 2007], consists of learning the parameters (i.e. the probabilities) of this BN. The parameters are learnt from the collected babbling data (e.g., Table 4.1) by using the maximum likelihood parameter estimation from the BNT toolbox [Murphy et al., 2001] for Matlab.

At this point the robot has obtained the BN representing a single object affordance, which it can now use to reason about its interactions with the environment. Section 4.4 will show how this model will be extended to a relational model by considering two object interactions and modeling with a PPL, which is the main contribution of this chapter.

4.3.2 Using the Affordance Model

Figure 3.1 showed the three possible uses of the affordance model, in which one of the three feature sets (O, A, E) can be computed by providing the other two. We will focus on the task of action prediction or recognition, where the robot observes O and E and has to *infer* A, the action that was performed. This is useful in imitation learning: the robot observes a human manipulating an object (and properties), observes the effects of the demonstration and tries to infer the action to imitate the effects in terms of its own action repertoire (obviously different from that of the human).

Action prediction is also a basic step in *planning*, as the robot knows the effects it wants to achieve and tries to find the sequence of actions needed. The experimental section will address this task—we will show how our approach performs action prediction in the context of single-action planning. In order to perform *action recognition*, we use the previously learned BN representing the affordance model and compute the *maximum a posteriori probability* (MAP) estimate: $\arg\max_A P(A|O,E) = \arg\max_A \frac{P(A,O,E)}{P(O,E)}$, with the observed values for O and E.

4.4 Relational Models for Affordances

In this section we describe our main contribution: the extension of the previously introduced one-object affordance models employing BNs to more general settings using a probabilistic relational model. In these, manipulation skills can involve multiple objects, object interactions occur while manipulating and different behaviours are required depending on the spatial relations between objects. These relations include e.g., *relative distance* between objects and their *angle of orientation* and *contact*.

Our multi-object setting requires first-order logic to capture—and generalize over —the (spatial) relations in the domain, probabilistic information to deal with uncertainty in perception and action and learning to induce the affordance models from interaction with the environment. Through the use of *variables*, i.e. place-holders for individual objects, in the relational representation, the models are able to generalize to arbitrary numbers of objects in the scene. That is, we derive a relational knowledge representation model only by taking into account the single and the two-object affordance model (which includes relational features between two objects). This model can then be applied in any multi-object scene, as shown in the experiments section (e.g., to be evaluated with 6 objects).

4.4.1 PPL Modelling

In order to build a probabilistic relational model for the affordance model we will make use of a PPL. We will first build a PPL program to model the single object affordance model. This can then later be extended with logical rules modelling the background knowledge we have about the setting, as well by modelling two-object interactions taking their spatial relations into consideration.

So we can now continue with **Step 1b** of our process by showing how an SRL model of single object affordances can be built. This single object model, or parts of this model, can be included in the overall relational affordance model for cases when there are no object interactions. The robot could equally learn to model these cases from babbling settings where the two objects do not interact, but this in general would require significantly more learning instances.

For the modelling of affordances in PPL for the task in this chapter, the main predicates we will use are presented in Table 4.2 below.

Table 4.2: Predicates used for affordance modelling

Predicate	Meaning
`shape(Obj, Shape)`	The shape of object `Obj` is `Shape`.
`dispMag(Obj, Val)`	The discretised displacement magnitude of object `Obj` is `Val`.
`dispOri(Obj, Val)`	The discretised displacement orientation of object `Obj` is `Val`.
`action(Obj, Type)`	The action on object `Obj` is `Type`. `Type` can be one of: `push`, `tap`, `grasp`.
`handMotion(ObjM, ObjS, Type)`	The type of hand motion for an action that manipulates object `ObjM`, which in turn interacts with object `ObjS` is `Type`. `Type` is one of the three action types.
`initRelDist(Obj1, Obj2, Val)`	The discretised initial relative distance between objects `Obj1` and `Obj2` is `Val`.
`initRelOri(Obj1, Obj2, Val)`	The discretised initial relative orientation between objects `Obj1` and `Obj2` is `Val`.
`contact(Obj1, Obj2, Val)`	If there is contact between objects `Obj1` and `Obj2` then `Val` is 1, otherwise it is 2.

We first show how to model the obtained single object affordance model represented by a BN as in Figure 4.6. For representation of, and inference using, the relational affordance model, we utilize the state-of-the-

art PPL ProbLog [De Raedt et al., 2007], with causal-probabilistic logic (CP-logic)[Vennekens et al., 2009] style syntax, introduced in Section 2.3.1.

There are many other PPLs available beside ProbLog, including for example BLOG [Milch et al., 2005], Church [Goodman et al., 2008], IBAL [Pfeffer, 2001], and PRISM [Sato and Kameya, 1997], among others. All of them would have been suitable for modelling the discrete relational affordance model shown in this chapter. However, only a few of these, such as BLOG and Church for example, would support modelling the continuous relational affordances introduced in the following chapters. Apart from DDCs, BLOG together with a particle filter for inference is the only PPL which could also be used for creating dynamic models, which could be used for modelling our relational affordance temporal domain from Chapter 6. However, ProbLog and DCs/DDCs have an easy-to-use syntax and are available in our group. In this thesis we are mostly interested in showcasing how relational affordances can be modelled with a PPL, and not concerned with investigating the performance of a particular PPL for a given task. Hence, to show of our approach, we will be using ProbLog for the discrete domains, and DCs for continuous domains. Investigating which PPLs yield better performance for the given tasks can be seen as future work.

We can first start by modelling the nodes that have no parents in the BN. Having learnt the parameters of the BN, we need to model the values these random variables can take and their respective probabilities.

For example, to model the shape of an object being randomly chosen from a set of two predefined shapes (i.e. the shape can take exclusively one of the two values with a probability of $\frac{1}{2}$) one would write the clause:

$$\tfrac{1}{2} :: \texttt{shape}(\texttt{cube}); \tfrac{1}{2} :: \texttt{shape}(\texttt{cylinder}) \leftarrow \texttt{true}.$$

We will generalise over the number of objects, by introducing variables for objects (e.g., `shape(Obj)` for the *shape* in the BN), in order to be able to later build a general multiple object PPL model from BN. The reason for this will become apparent when we introduce learning of two object interactions.

For example, we can generalize the previous example clause over objects:

$$\tfrac{1}{2} :: \texttt{shape}(\texttt{Obj}, \texttt{cube}); \tfrac{1}{2} :: \texttt{shape}(\texttt{Obj}, \texttt{cylinder}) \leftarrow \texttt{obj}(\texttt{Obj}).$$

where variable `Obj` is universally quantified over the set of all objects.

We want to have a general ProbLog model derived from the one (and later two) object affordances, which are modeled with BNs. As explained in Section 2.3.1, we can create an SRL program from any BN.

For example, using the BN model in Figure 4.6 with previously learned

parameters, part of the relationship between the action and the object displacement magnitude is modeled as:

$$0.03 :: \texttt{dispMag}(\texttt{Obj}, 1); 0.22 :: \texttt{dispMag}(\texttt{Obj}, 2); \\ 0.25 :: \texttt{dispMag}(\texttt{Obj}, 3); 0.5 :: \texttt{dispMag}(\texttt{Obj}, 4) \leftarrow \texttt{action}(\texttt{Obj}, \texttt{push}).$$

When an object is pushed, its displacement magnitude takes one of the 4 possible values with the specified probability.

Finally, we can also augment our model with logical rules to represent background knowledge, for example:

$$\texttt{dispMag}(\texttt{Obj}, 2) \leftarrow \texttt{obj}(\texttt{Obj}), \texttt{action}(\texttt{Obj}, \texttt{push}).$$

which models that the displacement magnitude of any object takes the value "2" when that object is pushed.

At this point we have defined a single object affordance model in a PPL, and we can use this model for inference. Several inference methods are available for computing the probabilities of a user's query. This means asking for the success probability $P(q|T)$ of a query q, which is the probability that q has a proof given the distribution over logic programs [De Raedt et al., 2007].

We mentioned that we learned the parameters of the affordance model BN using maximum likelihood parameter estimation. For a single or small predefined number of objects this can be done as in [Lopes et al., 2007, Montesano et al., 2008], but this approach for learning the parameters will not generalize over any number of objects or deal with partial observations (i.e. not all of O or E observed, e.g., faulty sensors). As presented in Section 2.3.1, ProbLog LFI could be used for this purpose.

If we already have the structure of the program, building the ProbLog LFI program is similar to building a ProbLog one, where instead of probabilities every fact has a $\texttt{t}(\_)$ prefix (e.g., $\texttt{t}(\_) :: \texttt{shape}(\texttt{cube})$ for the probability of the shape being a cube), indicating that the probability should be learnt. The program needs to also be supplied with a set of examples. For example, we can have two training examples, one in which the shape is a cube, and one in which it is not:

$$\texttt{example}(1). \\ \texttt{known}(1, \texttt{shape}(\texttt{cube}), \texttt{true}). \\ \texttt{example}(2). \\ \texttt{known}(2, \texttt{shape}(\texttt{cube}), \texttt{false}).$$

However, because of a lack of availability of ProbLog LFI for annotated

disjunction settings, and the fact that for our task we had full observations, this approach was not used for evaluation.

4.4.2 Learning Two Object Models

Having learnt single object affordances, which we modelled using the PPL ProbLog, we can now continue with learning from two-object babbling data.

To choose the settings for the two-object babbling stage, we place one object (the object the robot will act on) on the table in the middle of the action space for the respective action that the robot will execute. Then, we place a second object around this object at locations where the distance between the two objects on the x and y-coordinates is smaller than the hand movement distance during the action. Similarly, we vary the x and y-coordinates of this second object to generate the multiple babbling instances for that action. For each of the three actions, we generated over 50 babbling settings.

Just like in the case of single objects, the first step, namely **Step 2a** of our approach, is to learn a BN from the collected data.

In the two-object setting, we will define the *main* object to be the one the robot acts upon, and the *secondary* object the other object in the scene, which may interact with the main one through the robot's actions. Both are arguments of the robot's hand motion, and the action is defined over any secondary object:

$$\texttt{action}(\texttt{ObjMain}, \texttt{A}) \leftarrow \texttt{handMotion}(\texttt{ObjMain}, \texttt{ObjSec}, \texttt{A}).$$

For multi-object scenes we introduce, in addition to the O and E features detected for the single object case, two spatial relational object properties: the *relative distance* ($initRelDist$) between two objects and the *relative orientation* ($initRelOri$) of one with respect to the other, and one relative effect: whether there is *contact* between them. The robot first explores the action space for two objects as seen in Figure 4.7 (we ran about 600 scenarios) to learn an affordance model. An example of one such collected O, A, E set, for a *tap* action, is shown in Table 4.3.

From the data a grounded BN (structure and parameters) was learned in a similar manner as for the single object case in Section 4.3. We use this BN in ProbLog while generalizing over the number of objects by introducing variables for objects, resulting in the non-grounded BN in Figure 4.8. A non-grounded BN is a BN that would contain variables.

After modeling the two-object data with the help of a BN, we can move forward to **Step 2b** of our relational affordance modelling: building an SRL model that

Table 4.3: Collected O, A, E data for the tap action (discretised values)

Object Properties	Action	Effects
$shape_{O_{Main}} : square$ $shape_{O_{Sec}} : square$ $initRelDist_{O_{Main},O_{Sec}} : 2$ $initRelOri_{O_{Main},O_{Sec}} : WNW$	tap	$dispMag_{O_{Main}} : 2$ $displOri_{O_{Main}} : WSW$ $displMag_{O_{Sec}} : 1$ $displOri_{O_{Sec}} : WNW$ $contact_{O_{Main},O_{Sec}} : true$

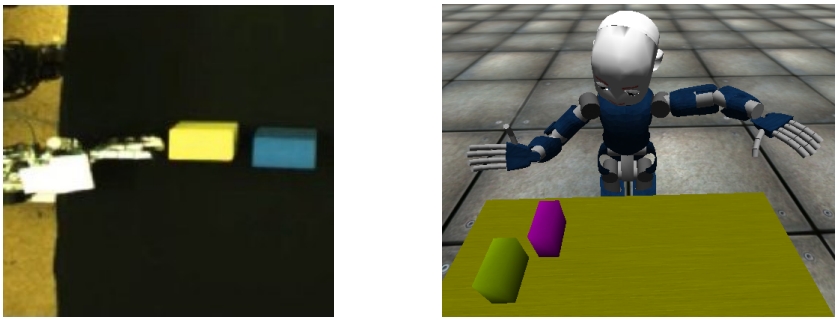


Figure 4.7: Real(l) and simulation(r) screenshot of the two object setting.

can be used in multiple object scenes.

Firstly, for creating models that capture two-object settings, we need to note that the single object BN can be reused by generalizing to two or more objects if there is no object interaction. Using this approach normally reduces the number of two-object babbling instances the robot needs, as the robot will not need to specifically explore two-object settings where there is no object interaction. In the displacement magnitude example, this is the case if the two objects are the furthest away (initial relative distance "4"). To model this, we will *manually* add the corresponding `initRelDist` predicate to the body of the clause we previously presented which defined the single object displacement magnitude. The effects on the main object are the same as in the one-object case, which can be modelled as:

$$
\begin{aligned}
&0.03 :: \texttt{dispMag}(\texttt{ObjMain}, 1); 0.22 :: \texttt{dispMag}(\texttt{ObjMain}, 2); \\
&\quad 0.25 :: \texttt{dispMag}(\texttt{ObjMain}, 3); 0.5 :: \texttt{dispMag}(\texttt{ObjMain}, 4) \\
&\quad \leftarrow \texttt{handMotion}(\texttt{ObjMain}, \texttt{ObjSec}, \texttt{push}), \\
&\quad \texttt{initRelDist}(\texttt{ObjMain}, \texttt{ObjSec}, 4).
\end{aligned}
$$

Once we have obtained the BN in Figure 4.8, as we have shown in Section 2.3.1,

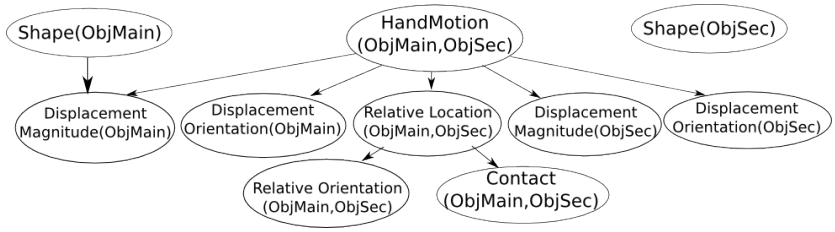


Figure 4.8: Non-grounded BN for two-object interaction.

we can model this BN with the help of ProbLog. For example, we can model the displacement orientation of the main object in case of a *push* action as:

$$0.044199 :: \texttt{dispOri}(\texttt{ObjMain}, 1); 0.381215 :: \texttt{dispOri}(\texttt{ObjMain}, 2); \\ 0.287293 :: \texttt{dispOri}(\texttt{ObjMain}, 3); 0.071823 :: \texttt{dispOri}(\texttt{ObjMain}, 4); \\ 0.082873 :: \texttt{dispOri}(\texttt{ObjMain}, 5); 0.088398 :: \texttt{dispOri}(\texttt{ObjMain}, 6); \\ 0.005525 :: \texttt{dispOri}(\texttt{ObjMain}, 7); 0.038674 :: \texttt{dispOri}(\texttt{ObjMain}, 8) \\ \leftarrow \texttt{action}(\texttt{ObjMain}, \texttt{push}).$$

Also from the BN in Figure 4.8, we would model contact between the objects when their initial distance is small (i.e., value 1) as follows:

$$0.8 :: \texttt{contact}(\texttt{ObjMain}, \texttt{ObjSec}, 1); 0.2 :: \texttt{contact}(\texttt{ObjMain}, \texttt{ObjSec}, 2) \\ \leftarrow \texttt{initRelDist}(\texttt{ObjMain}, \texttt{ObjSec}, 1).$$

All the rest of the BN can be modelled in a similar manner. So, at this point, we have built a generalised model based on two-object interactions.

We are now ready to go on to the final part of our modelling approach in PPL, namely the addition of logical rules for the respective setting of the robot. We will illustrate some of this knowledge the robot needs to achieve its manipulation task now.

Logical rules are added for defining background knowledge, which the robot would take longer to learn by itself otherwise. For example, for the cases when the objects are far away, the secondary object would not move (lowest displacement: "1"):

$$\texttt{dispMag}(\texttt{ObjSec}, 1) \leftarrow \texttt{handMotion}(\texttt{ObjMain}, \texttt{ObjSec}, \texttt{push}), \\ \texttt{initRelDist}(\texttt{ObjMain}, \texttt{ObjSec}, 4).$$

Logical rules can also be used to enforce constraints in the setting, e.g., if two objects are close together a grasp should not be performed (the secondary object would interfere with the hand). We can do this, for example, by ensuring that the *grasp* hand motion holds only in the case when the initial distance between

the two objects is not minimal (i.e., value 1). In ProbLog code, this can be coded as the following constraint:

```
actionConstraint(ObjMain, ObjSec, A) ←
    (A == grasp− >
    initRelDist(ObjMain, ObjSec, Dist), Dist ≠ 1; true
    ).
```

Then the predicate `actionConstraint` can be used in the body of the clause that defines `handMotion`. So if the constraint does not hold, `actionConstraint` will be false, and `handMotion` will be false.

Rules can also be added for enforcing the action space or encoding relations not caught by structure learning. While this was not used for our specific task in this section, one example can be that *push* actions can only be performed for objects closer than $20cm$.

We will now have a brief discussion on the balance between learning the whole model from data collected through a babbling stage, and using background knowledge in the form of logical rules for constructing parts of the model.

The advantage of introducing the logical rules is that we can significantly reduce the number of babbling instances needed for building the model. For example, let us consider the logical rule presented earlier that stated that when objects are far away (initial relative distance: "4"), then the secondary object would not move (lowest displacement: "1") during a push action. To learn this from babbling data, we would probably want to have the robot explore an environment with all combinations of two object types, and all possible initial relative orientations between the two objects. In our particular setting, given the chosen discretisations, this means at the very least a minimum of 24 babbling instances. Note that generally speaking, we would want more than one instance of a particular setting in order to generalise to a rule. It can be seen how the more complex an environment is, and the more object properties, actions, and effects are modelled, the more babbling instances would be needed, and so the use of logical rules significantly reduces the time the robot spends in the babbling phase.

Another aspect that needs to be considered when building a model using data from a babbling stage is the errors introduced by the perception system. As shown later in experiments in Section 6.6, perception success in a table-top multiple object environment can be 75% or lower. This means that modelling using logical rules can be more accurate, if we have detailed knowledge about the domain we want to model. For example, in our setting, we know that in a *push* action, the hand moves a distance of $4cm$. Therefore, secondary objects farther away will not be affected. However, during a babbling stage when using

our perception system, these far away objects could be erroneously detected as being closer, thus introducing errors in the data used to build the model.

The overall model can thus be built from a combination of expert knowledge and data gathered during the babbling stage. In this sense, we can also have an initial rough model of the setting, which can be adjusted with new babbling data, or by the addition of more specific rules. Learning a BN from a combination of knowledge and statistical data has been studied in research such as [Heckerman et al., 1995]. There the authors develop algorithms that take as input a user's prior knowledge, expressed mostly as a prior BN, together with obtained statistical data, and return an improved BN. Most recently, such approaches combining prior expert knowledge and statistical data has been investigated for example in the medical domain [Flores et al., 2011], where the authors develop a hybrid approach which incorporates prior expert information into the causal discovery process applied to a heart failure dataset. Therefore, combining both prior expert knowledge and data obtained during a babbling stage can be beneficial and improve the overall model.

However, in order to introduce these logical rules, expert knowledge about the respective domain that needs to be modelled is necessary. In the case of table-top object manipulation which is investigated here, this is easy as the expert knowledge consists of Newtonian physics and every-day common sense knowledge. In more complicated domains, an expert user will be required to code these logical rules about the specific domain needing to be modelled.

The new SRL model presented in this section generalizes over specific objects and can be applied in arbitrary scenes. In contrast to a standard BN, we do not have to encode all relations between all objects and generic rules are applied to arbitrary objects, so the number of parameters to be learned can be exponentially lower.

The full relational affordance model can be found in Appendix A. Please note that for presentation clarity, some of the constant and atom names were changed here, as well as some of the language details were left out.

4.4.3 ProbLog Inference for Action Recognition

The ProbLog model can now be used by the robot for probabilistic inference. Here we are interested in action prediction (finding A given O and E). This means calculating a MAP estimate: $\arg\max_{A,ObjMain} P(A_{ObjMain}|O,E)$. In practice one just needs to do inference using the ProbLog program to obtain the required probability. Another advantage is that if a sensor fails and a feature

value is missing, ProbLog is able to marginalize over the missing variables and find the required probabilities and predict A nonetheless.

The example below illustrates one such case of inference:

Detected O: shape(o1, rect), shape(o2, square), initRelDist(o1, o2, 1)
Desired E: dispMag(o1, 4), dispMag(o2, 2),
dispOri(o1, NNE), dispOri(o2, NNW), contact(o1, o2, 1)
Predicted A: push(o1): 7.7%, tap(o1): 0%, grasp(o1): 2.0%
push(o2): 90.1%, tap(o2): 0.2%, grasp(o2): 0.0%

In this case, the action predicted is a push on $o2$.

4.5 Evaluation and Results

We want to show that the SRL model obtained from two-object interactions can be used in a general multi-object setting with comparable results to a BN model trained specifically for that number of objects. This will also show that interactions between three or more objects need not be explicitly considered because the relevant dependencies can usually be captured by pairwise interactions, and any additional influence of these interactions is usually negligible.

We evaluate our approach in the context of single-action planning where the robot needs to pick the object to act on and the best action to match some required effects. This constitutes **Step 3** of our approach. By evaluating our approach, we want to find out:

(i) Can the robot pick the right object?

(ii) Can it pick the right action?

(iii) Can it pick the right action on the right object?

(iv) How does the SRL approach compare with the BN approach?

We use a six-object setting to investigate if our SRL model is able to generalize from two-object interactions. The objects are placed in front of the robot on the table. Three objects are always in the field of action of the robot, though the robot might not be able to perform all the actions on every object as this might violate some rules (e.g., action space for that action, interference with the hand from nearby objects). The other three objects are placed behind these and might interact with them when performing an action. All the objects are

randomly placed within certain margins and have a random shape. One such placement is shown in Figure 4.9(1). This is similar to the shelf setup, where the objects at the back are "in the shelf" while the ones in the front need to be arranged.

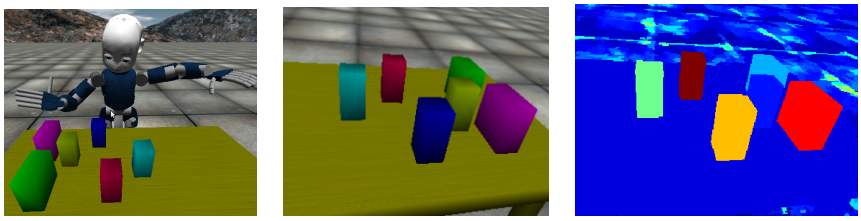


Figure 4.9: Six objects(1), left eye image(2) and its segmented objects(3).

In each setting setting, we execute all possible actions with the iCub to get real-world matching effects. Given these effects and the object properties, we predict the action and compare it against the ground truth action we performed.

Our SRL model consists of the *generalization of the two-object interactions* described by the BN in Figure 4.8 and the following rules (one of each type):

(i) *generalization from one object*: if the main and secondary object are far away, use the displacement orientation from the one-object case for the main object,

(ii) *background knowledge*: if the main and secondary object are far away, displacement of the secondary object is lowest,

(iii) *constraint enforcement*: grasp is only allowed if object distance is not lowest.

Here are examples of each of the above rules. For more details, please refer to Appendix A. An example of rule (i) is the following:

$$
\begin{aligned}
&0.000001 :: \texttt{dispOri}(\texttt{ObjMain}, 1); 0.343749 :: \texttt{dispOri}(\texttt{ObjMain}, 2);\\
&\quad 0.531250 :: \texttt{dispOri}(\texttt{ObjMain}, 3); 0.062500 :: \texttt{dispOri}(\texttt{ObjMain}, 4);\\
&\quad 0.031250 :: \texttt{dispOri}(\texttt{ObjMain}, 5); 0.000000 :: \texttt{dispOri}(\texttt{ObjMain}, 6);\\
&\quad 0.031249 :: \texttt{dispOri}(\texttt{ObjMain}, 7); 0.000001 :: \texttt{dispOri}(\texttt{ObjMain}, 8)\\
&\quad \leftarrow \texttt{handMotion}(\texttt{ObjMain}, \texttt{ObjSec}, 1),\\
&\quad \texttt{initRelDist}(\texttt{ObjMain}, \texttt{ObjSec}, 4).
\end{aligned}
$$

We will also need to add similar code for the other two actions (i.e., $\texttt{handMotion}(\texttt{ObjMain}, \texttt{ObjSec}, 2)$ and $\texttt{handMotion}(\texttt{ObjMain}, \texttt{ObjSec}, 3)$).

The code of rule (ii) is the following:

```
1.0 :: dispMag(ObjSec, 1) ← action(ObjMain, ObjSec, 1),
   initRelDist(ObjMain, ObjSec, 4).
```

The code of rule (iii) is:

```
handMotion(ObjMain, ObjSec, A)
   ← actionConstraint(ObjMain, ObjSec, A), ...
```

Experiments will show that just this can give good results. Adding more rules increases the prediction rate, but inference will take longer.

To investigate single-action planning, consisting of action and main object prediction, we ran 200 experiments against which we tested our SRL model. For the BN model we split the same data into two sets and used one for training and one for testing. We did this six times and averaged the results, since the results vary because the number of experiments is relatively small given the high number of nodes.

We do color segmentation, illustrated in Figure 4.9 (2, 3) for the left eye, and then 3D object localisation using stereo vision on the scene to find out the position of each object.

Every possible $a \in A$ is executed and recorded. For calculating the effects, the scene is segmented again, and we compute the displacement and its angle orientation. On a given image, the contact feature relies on the area of intersection between the convex hull of the two segmented areas. The intersection is normalized in two ways: (i) with respect to the minimum object area in order to remove image size dependencies and (ii) with respect to the distance of the farthest object in order to remove depth dependencies. The normalized contact feature is averaged over both cameras.

We use the O and E data values and the SRL model to predict the action and main object by calculating $\arg\max_{A,ObjMain} P(A_{ObjMain}|O, E)$ and compare them to the real action-object pair. Similarly, we obtain the best predicted object to act on by summing over all possible actions in the above formula, and the best predicted action by summing over all possible main objects. To have a baseline to compare our SRL approach to, we learn the six-object BN model. In this model, the action node has 9 possible values (3 actions for each of the 3 reachable objects), and we compute the MAP estimate to find out the predicted object-action duo. The comparison of the two approaches is shown in Table 4.4.

So, the robot can: (i) pick the right object in 74.5% of the trials, (ii) pick the right action in 68.5% of the trials, and (iii) pick the right action on the right object in 58% of the trials. These results are comparable to the BN approach,

Table 4.4: Action prediction in six-object scenarios.

	Prediction Task	Total exp.	Success	Pct.
ProbLog Model	$ObjMain$	200	149	74.5%
	A	200	137	68.5%
	$ObjMain$ and A	200	116	58%
BN Model (avg over 6 train/test sets)	$ObjMain$	100	67	67%
	A	100	69.2	69.2%
	$ObjMain$ and A	100	67	67%

but the BN would need to be retrained every time the setting changes.

As explained in more detail later on in Section 6.6, for the iCub robot the average action motor success is 85.15%, while average perception success is 82.03%. So both the motor action and perception are successful only in 69.85% of the cases. Since during our action prediction task, only one action is executed, and so the robot cannot recover from its mistakes, the action motor success rate of 85.15% also represents the theoretical success upper bound of the system.

Because each object-action combination is modeled by a different value in the action node of the BN and the relatively low number of training examples, the BN predicts the object-action duo and its individual components with almost the same rate. The approaches have comparable results; the SRL is slightly better at predicting the object to act on, while the BN is slightly better for the object-action duo. However, the SRL model can be used without being changed for any number of objects, while the BN needs to be learned again and can get very big (a six-object setting BN already has 65 nodes). In addition, the SRL enables transferring structural parts of the model (e.g., abstract action-effect rules) to similar domains with more, less or other types of objects. If we want improved accuracy tailored to a setting, the SRL model can also be trained with data from that setting using LFI.

We also looked at some learning statistics for our SRL model (summarized in Table 4.5): i) the confidence of the predictions (i.e. the value of $P(A|O, E)$) and (ii) the number of correct effects produced by an incorrect predicted action.

We see that the confidence of predicting an action is very high, while (as expected) that of the object-action duo is lower. When our prediction is wrong, executing the action still manages to produce about $\frac{2}{3}$ of the 27 effects correctly, sometimes a good compromise in complex scenarios.

There are several limitations in the evaluation step, which we will discuss briefly.

Table 4.5: Learning statistics of the SRL affordance model.

Prediction Result	Statistic	Pct.
Success	$ObjMain$ confidence	73.1%
	A confidence	90.1%
	$ObjMain$ and A confidence	62.6%
Failure	Correct effects	67.2%

The iCub robot with a fixed torso has a relatively small action space. This is the reason for the three objects in the front layer in the experiments, as this is the maximum number of objects of the mentioned sizes that can fit in the iCub action space, and also leaving a small space between them for the manipulation actions. Smaller objects could be used, but this will increase the effect uncertainty as the robot hand will be bigger than the manipulated objects. The actions being preprogrammed, we also need to be careful for objects on the table that might get in the way of the action execution. The orientation of the objects is also important. If the cubes are placed with their edge towards the robot instead of their sides, the effects of the actions will be different. We could also have the robot learn this from babbling examples, and then model this aspect as well, but since this considerably increases the complexity of the model and babbling data instances required, it will be considered as future work. Finally, another limitation of the system is the need for the user to know and model the background knowledge. While this might be easy in this particular table-top setting (e.g., objects far away do not interact when one of them is acted upon), it might be a hard task in more complex scenarios.

4.6 Related Work

The concept of relational affordances builds upon previous work in affordances, a concept introduced by J. J. Gibson [Gibson, 1979] and used to model world-robot interaction. More specifically, in this chapter we extended previous work on affordances in the context of imitation learning in [Lopes et al., 2007, Montesano et al., 2008] where an affordance model for a single object is learned with a BN in the context of a robot interaction game. Related work is also the use of the affordance models in the context of word-to-meaning association in [Krunic et al., 2009]. In order to extend the affordance model to the relational domain, this chapter built upon work in SRL, including [De Raedt, 2008, De Raedt and Kersting, 2008]. The ProbLog language which

was used for illustrating relational affordance models in this chapter is described extensively in [De Raedt et al., 2007, Gutmann et al., 2011a]. Finally, this work falls in the same area as that of probabilistic robotics [Thrun et al., 2005] and of providing robots with logic and probabilistic reasoning capabilities [Jain et al., 2009, Hertzberg and Chatila, 2008], as well as in the context of planning [Lang and Toussaint, 2009] and object-action complexes [Wörgötter et al., 2009].

There has also been recent research in the past couple of years that is related to our relational affordance work. Research on bootstrapping paired-object affordance learning with learned single-affordance features [Ugur et al., 2014] tackles the learning of complex multi-object affordances. Firstly, single object affordances are learnt as classifiers predicting effect categories given object features for the given actions, and secondly learning complex multi-object affordances is bootstrapped by using the previous object and affordance features. Research on recognizing object affordances in terms of spatio-temporal object-object relationships [Pieropan et al., 2014] presents a probabilistic framework that models the interaction between multiple objects in a scene. Pairwise interactions between objects are encoded in a spatio-temporal feature, which by the use of a kernel representation is embedded in a vector space which allows to define a metric comparing interactions of different temporal extent. A probabilistic model is created based on this metric, which can be used for extracting the affordances of individual objects based on the structure of their interaction. Finally, research on knowledge propagation and relation learning for predicting action effect [Szedmak et al., 2014] tackles the task of learning to predict the effects of actions applied to pairs of objects. Affordance predictions are propagated by exploiting the similarities among object properties, action parameters and effects. A Maximum Margin Multi Valued Regression approach is extended for learning paired-object affordances, and predicting the effects of actions applied on pairs of objects.

4.7 Conclusion and Future Work

We have presented an approach for robotic affordance model learning in a probabilistic relational setting. Moving to multi-object scenes requires expressive representation schemes to generalize over specific spatial configurations of objects and dealing with uncertainty and partial knowledge about the environment. We showed that a relational extension of the affordance models of two object interactions can be used for modeling a multi-object scene with success. We have showed both in a discrete and a continuous domain setting how relational affordance models can be learnt and used.

Future work will study imitation learning, involving recognizing low-level "atomic actions" of one or two objects in a multi-object scene using the learned models. We also want to use affordance models for *planning* of manipulation strategies, where a task consists of sequences of actions and the robot learns a high-level manipulation strategy. The long term goal is to go towards an autonomous shelf sorting capability of the humanoid robot, as presented in Section 4.1.

Chapter 5

Two-Arm Robots Models

This chapter[1] first introduces the learning of relational affordances in a continuous domain setting, where object properties and effects can be modelled by continuous distribution random variables. Then we will extend the concept of relational affordances in order to model a two-arm robot. A relational affordance model can first be learnt for one arm through a behavioural babbling stage, and then with the use of statistical relational learning, after constructing a symmetrical model for the other arm, two-arm manipulation actions can be modelled, where the arms can act sequentially or simultaneously. Furthermore, background knowledge about the environment and objects to be manipulated will also be modelled.

5.1 Introduction

We have seen in the previous chapters how the use of SRL can be used to help model the world-robot interaction. This is advantageous, since SRL can model both the uncertainty in the physical world, such as noisy sensors, noisy camera image streams, and noisy physical actuators, and the logical representations used for dealing with higher level knowledge for reasoning and planning in the environment.

Recent advances in robotics have led to an increase in the number and capabilities of more advanced humanoid robots, such as the PR2, iCub, NAO, and other robotic platforms. One characteristic of these humanoid robots is having two

[1] This chapter is based on work previously published in [Moldovan and De Raedt, 2014a]

symmetrical arms with which they can manipulate their environment. The goal is for humanoid robots to manipulate objects in a household environment, such as a kitchen or a living room. In these cases, the robot should also consider background knowledge about the environment and objects to be manipulated, knowledge which a human would posses about the environment and task. For example, objects that are close together are likely to interact during manipulation, or that several given types of objects are used together to achieve a task (e.g., a fork and a knife). SRL is well suited to model background knowledge with the help of logical rules, while symmetries and generalisations are easily handled by the use of logic variables. Finally, SRL can be used to model the uncertainties in the task, sensing, or actuators, with the help of probabilities.

5.1.1 Problem Statement and Approach

In our two-arm robot manipulation scenario, we will tackle a table-top object manipulation scenario with multiple objects that can interact with one another during robot manipulation, e.g., in Figure 5.1. The task of the robot in this setting is to place the green bar, which it cannot reach directly, at the target location. It can achieve this by pushing simultaneously the two red objects with its two arms.

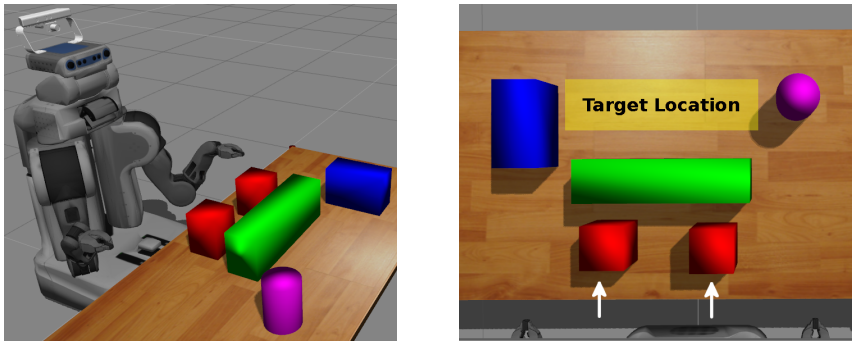


Figure 5.1: Table-top scenario with two-arm actions for object placement.

Note that the use of both arms, whose actions can be either simultaneous or sequential, enables the robot to perform tasks not possible by the use of only one (e.g., just pushing one of the red objects would not get the bar to the goal). By learning and using a two-arm relational affordance model, the robot can perform these even though it had never explored these settings during its one-arm behavioural babbling phase. The relational affordance model defines

a joint probability distribution over O, A, E, and it will be defined by a PPL program.

To tackle such object manipulation scenarios, three tasks need to be solved (two learning tasks and an inference task):

- Task (i) is learning a one-arm continuous relational affordance model: *given:* a) a set of corresponding O, A, E values collected from exploratory one-arm action executions in two-object environments, *find:* b) a continuous setting relational affordance model of one-arm actions
- Task (ii) is learning a two-arm model: *given:* a) the one-arm affordance model learnt in Task (i), and b) a set of background rules constraining two-arm actions (e.g., arms act on the same or on interacting objects), *find:* c) a relational affordance model of two-arm actions. Note that the robot can learn new affordances for objects for its two arms that were not possible with single arm actions.
- Task (iii) is the inference task used to evaluate our model: *given:* a) the two-arm affordance model learnt in Task (ii), b) an initial scene from which the set of object properties values O can be extracted, and c) a target goal, given as E, *find:* d) the best action to execute to reach the goal, given by: $\arg\max_A P(A|O, E)$.

To solve these tasks, we propose a pipeline, as shown in Figure 5.2, with the following steps:

- 1a) learn a Linear Continuous Gaussian (LCG) Bayesian Network (BN) from the exploratory data,
- 1b) from the LCG model, build a one-arm continuous domain relational affordance model in a PPL,
- 2) generalise the one-arm model to a two-arm model, and
- 3) perform action prediction using the two-arm model.

We will first show how a relational affordance model can be learnt in a continuous domain setting, which is Task (i) of our approach. Together, steps 1a and 1b will solve Task (i). The other two steps solve Tasks (ii) and (iii) respectively.

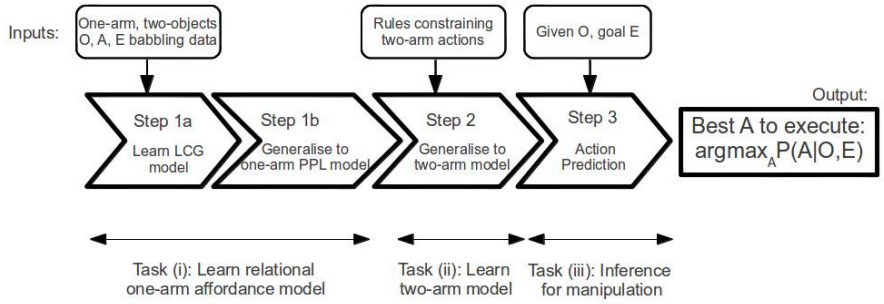


Figure 5.2: Pipeline for table-top two-arm object manipulation.

5.1.2 Contributions and Outline

The main contributions of this chapter are: i) extending the relational affordance models introduced in the previous chapter to a continuous domain setting, and ii) using SRL methods to create a relational affordance model for two-arm actions, for settings where these can be approximated by a combination of the two single-arm actions composing them. The arms may act simultaneously or sequentially, and the robot is given background knowledge about possible actions in its environment. Few studies have investigated two-arm manipulation, and our contribution is to use SRL to generalise and build a higher-level model for a set of two-arm actions settings in a household environment.

We continue presenting related work in Section 5.2. The extension of relational affordance models to the continuous domain, (i.e., Steps 1a and 1b in our pipeline) is shown in Section 5.3. Section 5.4 describes Step 2, our relational affordance model of two-arm actions. Section 5.5 presents experimental results, obtained by Step 3. We conclude in Section 5.6.

5.2 Related Work

Related work includes research on tool use for robots, such as presented in [Brown and Sammut, 2013], which learns the tool affordances of an object from a human demonstration together with a set of robot experimentations based on an inductive logic programming algorithm. Research on behaviour-grounded tool affordances such as [Stoytchev, 2005, Sinapov and Stoytchev, 2007] provides algorithms for the robot to learn the effects of its actions with given tools on other objects.

However, all these involve robots with only one end-effector. There are very few studies on two-arm robots. Among this, there is research on learning, representing and generalising a task which presents a programming-by-demonstration framework for extracting and generalising knowledge about a given task [Calinon et al., 2007], and similarly a programming-by-demonstration framework for dual-arm manipulation tasks [Zöllner et al., 2004]. There is also research on motion planning for dual-arm manipulation and re-grasping tasks [Vahrenkamp et al., 2009]. However, none of these use the concept of affordances, model or generalise over relations and interactions between manipulated objects and other objects in the environment, or build a two-arm manipulation model from a one arm affordance model obtained by environment experimentation and the use of background knowledge.

This chapter also proposes to find one solution to the motor equivalence problem for two-arm actions. Motor equivalence in robotics was inspired from physiology research, where Nikolai Bernstein stated that there are multiple ways for animals to perform a movement that achieves a certain goal [Bernstein, 1967].

5.3 Learning Relational Affordances in a Continuous Domain Setting

The relational affordance model presented in the previous chapter dealt with discretised data only. In realistic scenarios, the robot has to deal with continuous domain inputs and outputs, and to perform its inference accordingly. For this purpose, for the robot to tackle real-world manipulation settings we need to extend the previous relational affordance model to a continuous domain. We will present here how to learn relational affordances in a continuous domain, which will be used in all the complex manipulation tasks which will be presented later, including two-arm actions, multiple-action planning and object search. Learning relational affordance models in a continuous domain setting constitutes Steps 1a and 1b of our pipeline, which together achieve Task (i) of our problem setting.

5.3.1 Affordance Scenario Setting

We now describe more precisely our setting, introduced in Figure 5.1. In our setting, we employ the *PR2 humanoid robot* in the Gazebo simulator, in contrast with Chapter 4 which used the iCub simulator. We use both arms of the PR2 robot with the *arm_navigation* stack. Arm actions are performed by sending the arm to a goal location in Cartesian space, using inverse kinematics in order

to plan a trajectory for the arm to reach a position as close as possible to the goal within a *2cm* tolerance. An object can be acted upon by both arms, by one arm but not the other, or it can be completely out of the reach of the robot.

We now describe the variables of our affordance model that we will use for the continuous domain setting. They are illustrated in Figure 5.3, with the position of the objects before (l) and after (r) an action (tap) execution.

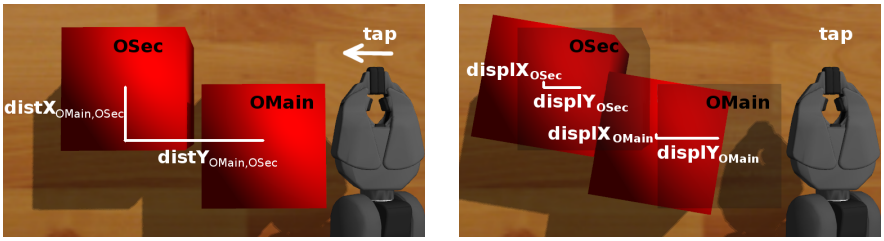


Figure 5.3: Relational O before (l), and E after the action execution (r).

The set of object properties O consists of the following: *shape*, and (the relational properties) relative distances along the x-axis ($distX$) and y-axis ($distY$) between the objects (in cm). As opposed to the relational affordance model introduced for the discrete case, we use a Cartesian coordinate system instead of a polar one, which facilitates modelling in a continuous domain setting (using Gaussian distributions for distances and displacements). We use four object shapes, the two previously used for the discrete case, denoted here by *cube* and *prism* (previous notation for the discrete case was *square* and *rect* respectively), and two additional ones: a *cylinder*, to increase object interaction variation, and a long *bar*, for two-arm manipulation. Properties $distX$ and $distY$ are measured centre-to-centre as in Figure 5.3(l).

Actions A are the two basic single arm core motor actions: *tap* (right-to-left hand movement for the right arm, left-to-right for the left) and *push* (away movement for both arms). Actions move the arm over a preprogrammed distance and orientation. This preprogrammed distance was chosen to be $7.5cm$, which is half the smallest of the two x or y-axes sizes of each object, which was $15cm$. For all the shapes except *bar*, the action is executed in the middle of the object. For *bar*, each arm acts towards its end of the object. Each arm action needs to be able to be performed on its own. The robot will first explore its environment in the babbling phase with only one arm (the right arm), and then to achieve its task it will execute two actions, one with each arm, which can be either simultaneous or sequential.

Effects E corresponds to differences in object attributes before and after the action is performed. We use the displacements along the x-axis ($displX$) and

y-axis ($displY$) of the centre of each object. These are shown in Figure 5.3(r), which overlays the initial positions of the objects over their final positions.

As in our setting the robot is not mobile and each arm has a specific action range, each $a_i \in A$ can be performed when an object is located in a specific action space. The action space is learnt during the babbling phase. If the exploratory arm action on an object fails because no inverse kinematics solution was found, then that object is not in that arm's action space. We will show later how any spatial constraints, such as action space, can be modelled with logical rules.

5.3.2 Learning a Linear Conditional Gaussian BN

Just like in the single object case, we will start by learning a BN from the data (corresponding O, A, E values) obtained from a behavioural babbling stage. This constitutes **Step 1a** of our pipeline. The behavioural babbling state is performed with the right-arm only. Pairs of objects are placed in front of the robot at various positions. For choosing these babbling settings we use a similar approach as that for one-arm actions presented in Chapter 4. We place one object in the centre of the action space for the respective action, and we place the second object at varying x and y-coordinates around this first object such that the distance between the objects is smaller than the hand motion distance.

The robot executes one of its actions A on one object (named: main object, O_{Main}). O_{Main} may interact with the other object (secondary object, O_{Sec}) causing it to also move. Figure 5.3 shows such a setting. The robot executed 300 such exploratory actions, obtaining 300 sets of O, A, E values. One such set, for the action shown in Figure 5.3, is shown in Table 5.1.

Table 5.1: Collected O, A, E data for the tap action in Figure 5.3

Object Properties	**Action**	**Effects**
$shape_{O_{Main}} : cube$ $shape_{O_{Sec}} : cube$ $distX_{O_{Main},O_{Sec}} : 7.00cm$ $distY_{O_{Main},O_{Sec}} : 17.00cm$	tap	$displX_{O_{Main}} : 0.40cm$ $displY_{O_{Main}} : 7.23cm$ $displX_{O_{Sec}} : 0.39cm$ $displY_{O_{Sec}} : 4.38cm$

Once we have collected the data, we learn a *Linear Continuous Gaussian (LCG) Bayesian Network* (BN) [Kjærulff and Madsen, 2005]. This LCG BN models a single-action step, and it specifically models right-arm actions. We will show later how, with the help of PPL modelling, we can extend the model to two-arms,

and to two-actions steps. In our setting $displX$, $displY$, $distX$ and $distY$ can be approximated by conditional Gaussian distributions over the short distances over which objects interact. We show later how to enforce these distances by adding logical rules. Later experiments for two-arm scenarios will show that this approximation is better than the discretisation presented in the previous chapter.

The LCG model of our setting is shown in Figure 5.4, where discrete random variables are represented by a single ellipse, and continuous ones by a double ellipse. $displX_{O_{Main}}$ and $displY_{O_{Main}}$ only depend on A and the object $shape$ since the hand is moved over a preprogrammed distance (with a given tolerance). $displX_{O_{Sec}}$ and $displY_{O_{Sec}}$ depend on both the relative distance O_{Sec} is away from O_{Main} and the shapes of both objects.

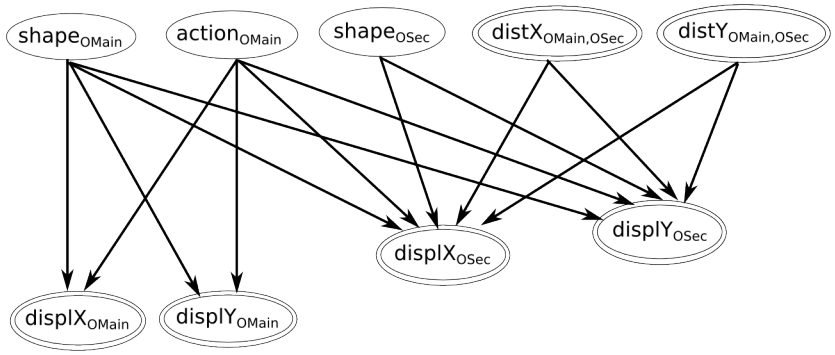


Figure 5.4: LCG BN model for two-object interaction

The LCG parameters are learnt from the collected babbling data (e.g., Table 5.1) by using the maximum likelihood parameter estimation from the BNT toolbox [Murphy et al., 2001] for Matlab. *E.g.*, during our *tap* action for two interacting cubes (as in Figure 5.3), the displacement of O_{Sec} on the y-axis is (in cm):

$$\mathcal{N}(19.92 - 0.05 * distX_{O_{Main},O_{Sec}} - 0.86 * distY_{O_{Main},O_{Sec}}, 0.17). \qquad (5.1)$$

Intuitively this makes sense: the second cube is moved along, so we expect the learnt coefficient of $distY$ to be close to -1, but also to depend a little bit on $distX$ if the objects are not aligned, as in Figure 5.3. Also intuitively, the mean coefficient generally depends on the widths of the shapes ($15cm$ for cubes), as well as the preprogrammed action distance.

5.3.3 PPL Modelling

We will now continue with **Step 1b** of our pipeline, modelling relational affordances in a PPL. In the previous chapter relational affordances were modelled using the PPL ProbLog. Here, since we deal with continuous distribution random variables, modelled by normal distributions, we use our new state-of-the-art PPL Distributional Clauses (DCs) [Gutmann et al., 2011b], a continuous extension of ProbLog.

For the modelling of affordances in PPL for the task in this chapter, the main predicates we will use are presented in Table 5.2 below.

Table 5.2: Predicates used for affordance modelling

Predicate	Meaning
`shape(Obj, Shape)`	The shape of object `Obj` is `Shape`.
`distX(Obj1, Obj2, T)`	Distribution of the relative x-axis distance between objects `Obj1` and `Obj2` at time-step `T`.
`distY(Obj1, Obj2, T)`	Distribution of the relative x-axis distance between objects `Obj1` and `Obj2` at time-step `T`.
`displX(Obj, Arm, T)`	Distribution of the x-axis displacement of object `Obj` due to an action with arm `Arm` at time-step `T`. `Arm` is one of `left` or `right`.
`displY(Obj, Arm, T)`	Distribution of the y-axis displacement of object `Obj` due to an action with arm `Arm` at time-step `T`.
`dX(Obj, D, T)`	The overall x-axis displacement of object `Obj` due to all actions at time-step `T` is `D`.
`dY(Obj, D, T)`	The overall y-axis displacement of object `Obj` due to all actions at time-step `T` is `D`.
`action(Type, Arm, Obj)`	The type of the action on object `Obj` with arm `Arm` is `Type`. `Type` can be `push` or `tap`.
`approx_ok(A, ObjM, ObjS, DX, DY)`	True if the Gaussian approximation for the action effect holds for the action `A` on main object `ObjM` and with secondary object `ObjS` when the x-axis distance between the objects is `DX` and the y-axis distance between the objects is `DY`.
`twoArmA(`$\mathtt{A_L}$`, `$\mathtt{A_R}$`, `$\mathtt{O_L}$`, `$\mathtt{O_R}$`)`	The robot action for the left arm is of type $\mathtt{A_L}$ on object $\mathtt{O_L}$, and for the right arm is of type $\mathtt{A_R}$ on object $\mathtt{O_R}$.
`actCheck(`$\mathtt{O_L}$`, `$\mathtt{O_R}$`)`	True if two-arm action with left arm on object $\mathtt{O_L}$ and right arm on object $\mathtt{O_R}$ is allowed.

We can now proceed to model the LCG using DCs to generalise to a relational affordance model. We generalise over the number of objects as before, by introducing variables for objects (e.g., $\texttt{displX}(\texttt{ObjMain})$ for the $displX_{O_{Main}}$ in the LCG), and so build a general multiple object PPL model from the two-object LCG BN. We illustrate the modelling with examples.

We first model the shape of an object being randomly chosen from our set of 4 shapes, each with 25% probability:

$$\texttt{shape}(\texttt{Obj}) \sim \texttt{finite}([\tfrac{1}{4} : \texttt{cube}, \tfrac{1}{4} : \texttt{prism}, \tfrac{1}{4} : \texttt{bar}, \tfrac{1}{4} : \texttt{cyl}]) \leftarrow \texttt{obj}(\texttt{Obj}).$$

where variable `Obj` universally quantified over the set of all objects.

Now we can model the LCG from Figure 5.4 with the learnt parameters. For example, to transform the LCG Equation 5.1 in DCs, one writes:

$$\begin{aligned}&\texttt{displY}(\texttt{ObjSec}) \sim \texttt{gaussian}(\texttt{Mu}, 0.17) \leftarrow \texttt{action}(\texttt{tap}, \texttt{ObjMain}),\\ &\quad \simeq(\texttt{shape}(\texttt{ObjMain})) = \texttt{cube}, \simeq(\texttt{shape}(\texttt{ObjSec})) = \texttt{cube},\\ &\quad \simeq(\texttt{distX}(\texttt{ObjMain}, \texttt{ObjSec})) = \texttt{DX}, \simeq(\texttt{distY}(\texttt{ObjMain}, \texttt{ObjSec})) = \texttt{DY},\\ &\quad \texttt{Mu is } 19.92 - 0.05 * \texttt{DX} - 0.86 * \texttt{DY}.\end{aligned}$$

meaning for a tap action, if the two shapes are cubes, `displY` of `ObjSec` is distributed according to a Gaussian with mean given by Mu.

We can use definite clauses to model that the above Gaussian approximation holds only over small distances ($10cm$ on the motion axis and while there is overlap on the orthogonal axis), while over big distances there will be no effect on `ObjSec`. For our two cubes running example: The distances `distX` and `distY` will be later given as evidence.

$$\begin{aligned}&\texttt{approx\_ok}(\texttt{tap}, \texttt{cube}, \texttt{cube}, \texttt{DX}, \texttt{DY}) \leftarrow \texttt{DY} > 15, \texttt{DY} < 25,\\ &\quad \texttt{DX} > -15, \texttt{DX} < 15.\end{aligned}$$

where $15cm$ is the smallest centre-to-centre distance between two cubes. We then just need to add $\texttt{approx\_ok}(\texttt{tap}, \texttt{cube}, \texttt{cube}, \texttt{DX}, \texttt{DY})$ to the body of the DC clause defining `displY` above. Similar rules can be added to enforce the action space.

At this point we have all the tools to fully model the relational affordance model with the parameters learnt as in Section 5.3.2. Once the program is defined, the inference algorithm based on sampling from [Gutmann et al., 2011b] or [Nitti et al., 2013] is used to compute the probability of a user's query.

For example, assuming two cubes o_1 and o_2, one can ask for the probability of the y-axis displacement of o_2 being greater than $3cm$ given some distance between o_1 and o_2. For this, we need to compute:
$P(displY(o_2) > 3 | action(tap, o_1), distX(o_1, o_2) = 4, distY(o_1, o_2,) = 2)$.

5.4 Modelling Affordances for Two-Arm Robots

Having obtained the relational affordance model for the right arm, we can now proceed to **Step 2**, creating the two-arm model. For this, we need to create first the model for the left arm. Given the symmetry of the robot, the model for the left arm is equivalent to the model for the right arm mirrored through the plane perpendicular to the table that passes through the centre of the robot. For our model (and Gazebo framework), in the left-arm model all the x-axis values are the same as for the right-arm model, but the y-axis values are the negative of their right-arm equivalent.

In our DC framework, the *displX* and *displY* random variables for left and right arm actions need to be defined by different probability distributions. So, we need to add an extra term to our `displX` and `displY` atoms to signify the arm performing the action. At this point, we can automatically generate the PPL code for the left arm. For our running example, the equivalent code for Equation 5.1 for the left arm:

$$\begin{aligned}&\texttt{displY}(\texttt{ObjSec}, \texttt{left}) \sim \texttt{gaussian}(\texttt{Mu}, 0.17)\\ &\quad \leftarrow \texttt{action}(\texttt{tap}, \texttt{left}, \texttt{ObjMain}),\\ &\quad \simeq(\texttt{shape}(\texttt{ObjMain})) = \texttt{cube}, \simeq(\texttt{shape}(\texttt{ObjSec})) = \texttt{cube},\\ &\quad \simeq(\texttt{distX}(\texttt{ObjMain}, \texttt{ObjSec})) = \texttt{DX}, \simeq(\texttt{distY}(\texttt{ObjMain}, \texttt{ObjSec})) = \texttt{DY},\\ &\quad \texttt{Mu is } 19.92 - 0.05 * \texttt{DX} - 0.86 * \texttt{DY}.\end{aligned}$$

Now we are ready to model two-arm actions, which we will define with the atom $\texttt{twoArmA}(\texttt{A}_\texttt{L}, \texttt{O}_\texttt{L}, \texttt{A}_\texttt{R}, \texttt{O}_\texttt{R})$, where $\texttt{A}_\texttt{L}$ and $\texttt{A}_\texttt{R}$ signify the left and right arm actions, and $\texttt{O}_\texttt{L}$ and $\texttt{O}_\texttt{R}$ the objects the arms act on:

$$\texttt{twoArmA}(\texttt{A}_\texttt{L}, \texttt{O}_\texttt{L}, \texttt{A}_\texttt{R}, \texttt{O}_\texttt{R}) \leftarrow \texttt{action}(\texttt{A}_\texttt{L}, \texttt{left}, \texttt{O}_\texttt{L}), \texttt{action}(\texttt{A}_\texttt{R}, \texttt{right}, \texttt{O}_\texttt{R}).$$

5.4.1 Adding Task Constraints in Environment

In a normal household environment there are many objects present, so it becomes infeasible for a robot to infer the success probability of acting on each of them when given a task. In our relational affordance model, we want to narrow down the state space search of the robot. So we can add constraints to our two-arm SRL model, which can be done by adding logical rules.

We first want to define some spatial constraints by defining the atom $\texttt{actCheck}(\texttt{O}_\texttt{L}, \texttt{O}_\texttt{R})$, which we will use in selecting the actions, and which only holds when our constraints on the objects acted upon by the left and right arm ($\texttt{O}_\texttt{L}$ and $\texttt{O}_\texttt{R}$ respectively) are met. In the evaluation of our approach, we use only the constraint that if the left and right arm act on two different objects, the

left arm should not act on an object more to the right than the one acted on by the right arm (i.e., y-distance between $\mathtt{O_L}$ and $\mathtt{O_R}$ positive):

$$\mathtt{actCheck}(\mathtt{O_L}, \mathtt{O_R}) \leftarrow \mathtt{O_L} \neq \mathtt{O_R}, \simeq(\mathtt{distY}(\mathtt{O_L}, \mathtt{O_R})) > 0.$$

More rules could be added to the body of the actCheck clause (e.g., one object should not be right behind the other) if needed by the specific task to be modelled.

Generally speaking, we want our relational affordance model to allow modelling of household environments, where objects are intended for human use and the robot tasks are similar to human tasks. We want to show how our model can be augmented for this type of environments by adding logical rules. In these environments, usually two-arm human actions fall into one of the following three categories:

1. Both arms act on the same object
2. One arm acts on an object, while the other acts on an object that interacts with the first object
3. The two arms act on two different objects that both interact with the same third object

The first type of two-arm action corresponds for example to unscrewing a bottle cap, or carrying around a tray. The second one corresponds with tool use by one arm, similar with [Brown and Sammut, 2013, Stoytchev, 2005] while the other arm supports or manipulates the object, e.g., hammering a nail. Lastly, the third type corresponds with both arms using tools to act on the same object, e.g., using a fork and knife on food.

We can do this by building upon our previous definition of `actCheck`. We can now define `actCheck` by three clauses to represent the three two-arm action categories presented above. At least one of the clauses needs to be true in order for `actCheck` to be true. For example, in our particular setting, we will model these clauses as follows:

$$\begin{aligned}
&\mathtt{actCheck}(\mathtt{Obj}, \mathtt{Obj}).\\
&\mathtt{actCheck}(\mathtt{Tool}, \mathtt{Obj}) \leftarrow \mathtt{Tool} \neq \mathtt{Obj}, \simeq(\mathtt{distY}(\mathtt{Tool}, \mathtt{Obj})) > 0,\\
&\quad \simeq(\mathtt{distY}(\mathtt{Tool}, \mathtt{Obj})) < 23.\\
&\mathtt{actCheck}(\mathtt{Tool1}, \mathtt{Tool2}) \leftarrow \mathtt{Tool1} \neq \mathtt{Tool2},\\
&\quad \simeq(\mathtt{distY}(\mathtt{Tool1}, \mathtt{Tool2})) > 0,\\
&\quad \simeq(\mathtt{distY}(\mathtt{Tool1}, \mathtt{Tool2})) < 30, \mathtt{shape}(_, \mathtt{bar}).
\end{aligned}$$

In our setting we modelled that two objects can interact (one being the tool) if their centre-to-centre y-axis distance is less than $23cm$. Given our object

shapes, this is the maximum distance separating the centres with the objects still being able to interact. If we use two tools, then the maximum distance is $30cm$, which is the length of the *bar* with which two tools can potentially interact. Different other rules can also be defined for different settings.

The use of these rules will restrict the search space of the robot when doing inference. Note that depending on the environment, different other background rules can be added as well (e.g., for object search, many objects are manipulated and moved away with both hands).

5.4.2 Sequential or Simultaneous Use of Arms

We finally need to model the effects of both arms acting on the environment. Note that the left and right arm actions can take place simultaneously (e.g., carrying a tray), or sequentially (e.g., one arm picks up an object and passes it to the other arm for manipulation). For this, we need again to add an extra term T (with values 1 or 2) to our atoms signifying the time-step. Furthermore, we model the overall displacement (`dX` and `dY`) of an object during a time-step as the sum of the displacements caused by left and right arm actions during that time-step. For example, along the y-axis:

$$\begin{aligned}&\texttt{dY}(\texttt{Obj},\texttt{D},\texttt{T}) \leftarrow \texttt{D is } \simeq(\texttt{displY}(\texttt{Obj},\texttt{left},\texttt{T}))\\&\quad + \simeq(\texttt{displY}(\texttt{Obj},\texttt{right},\texttt{T})).\end{aligned}$$

Note that for one or both arms, the `displY` distributions might not be defined (if the object is not manipulated or does not interact with one that is). So we need to also add the three definite clauses for `dY` corresponding to these cases. These are the following:

$$\begin{aligned}&\texttt{dY}(\texttt{Obj},\texttt{D},\texttt{T}) \leftarrow \texttt{not}(\texttt{displY}(\texttt{Obj},\texttt{right},\texttt{T})),\\&\quad \texttt{D is } \simeq(\texttt{displY}(\texttt{Obj},\texttt{left},\texttt{T})).\\&\texttt{dY}(\texttt{Obj},\texttt{D},\texttt{T}) \leftarrow \texttt{not}(\texttt{displY}(\texttt{Obj},\texttt{left},\texttt{T})),\\&\quad \texttt{D is } \simeq(\texttt{displY}(\texttt{Obj},\texttt{right},\texttt{T})).\\&\texttt{dY}(\texttt{Obj},\texttt{0},\texttt{T}) \leftarrow \texttt{not}(\texttt{displY}(\texttt{Obj},\texttt{right},\texttt{T})),\texttt{not}(\texttt{displY}(\texttt{Obj},\texttt{left},\texttt{T})).\end{aligned}$$

Similar rules are defined for the x-axis displacement.

At this point, we can already do action prediction for two-arm simultaneous actions A_L and A_R, since this lasts only one time-step. For this, given our overall DCs model for the two arms, we just need to calculate the MAP estimate: $\arg\max_{A_L,O_L,A_R,O_R} P(twoArmA(A_L,O_L,A_R,O_R)|O,E)$ by doing inference in our PPL program.

For sequential actions, we have a two-step planning problem, which needs to be

modelled in our probabilistic setting. There are several ways of achieving this, we chose to model it in our PPL with a probabilistic STRIPS formalism as in [Zettlemoyer et al., 2005, Pasula et al., 2004], where in comparison to classical STRIPS, each action has several outcomes, each associated with a probability that it might occur.

For illustration, consider the example with two cubes shown in Figure 5.3, which we expand by considering sequential two-arm actions. We will refer to the right cube as $\mathtt{o_r}$ and the left as $\mathtt{o_l}$. The first action will be the right-hand tap on $\mathtt{o_r}$ as in Figure 5.3. We assume the second action will be a left-hand push on $\mathtt{o_l}$ for $7cm$, in which case the two objects will not interact during the second action.

The first task consists of defining the states. In our particular table-top setting, we can define the state the objects are in by their defined (affordance) object properties. So, a state will consist of a conjunction of grounded `shape`, `distX` and `distY` atoms.

In our example from Figure 5.3, with collected data O, A, E as in Table 5.1, the initial state S_0 in STRIPS notation is:

$\mathtt{shape}(\mathtt{o_r}, \mathtt{cube}), \mathtt{shape}(\mathtt{o_l}, \mathtt{cube}), \mathtt{distX}(\mathtt{o_r}, \mathtt{o_l}, 7), \mathtt{distY}(\mathtt{o_r}, \mathtt{o_l}, 17).$

The left column of Table 5.3 shows S_0.

Table 5.3: Example states for two-arm actions.

S_0	S_1	S_2
$shape(o_r, cube)$	$shape(o_r, cube)$	$shape(o_r, cube)$
$shape(o_l, cube)$	$shape(o_l, cube)$	$shape(o_l, cube)$
$distX(o_r, o_l, 7)$	$distX(o_r, o_l, 6.99)$	$distX(o_r, o_l, 13.99)$
$distY(o_r, o_l, 17)$	$distY(o_r, o_l, 14.15)$	$distY(o_r, o_l, 14.15)$

To finish modelling the states, we are left with defining the relation between our `displX` and `displY` object displacements (affordance) effects and our state literals. For this we observe that the x-axis and y-axis distances between objects in the next state can be defined in terms of the ones in the previous state and the object displacements. For example:

$$\mathtt{distY}(\mathtt{O_1}, \mathtt{O_2}, \mathtt{D}, \mathtt{T}) \leftarrow \mathtt{PrevT\ is\ T} - 1, \mathtt{distY}(\mathtt{O_1}, \mathtt{O_2}, \mathtt{PrevD}, \mathtt{PrevT}),$$
$$\mathtt{dY}(\mathtt{O_1}, \mathtt{Y_1}, \mathtt{T}), \mathtt{dY}(\mathtt{O_2}, \mathtt{Y_2}, \mathtt{T}), \mathtt{D\ is\ PrevD} + \mathtt{Y_2} - \mathtt{Y_1}.$$

so the overall state model can be represented by using the shapes and relative distances atoms.

For example, to compute `distY` at S_1 between $\mathtt{o_r}$ and $\mathtt{o_l}$: $17{+}4.38{-}7.23 = 14.15$.

This can be seen in the second column of Table 5.3. The rest of S_1, and the final goal state S_2 after the two actions can be computed similarly.

Using the clause above and our DCs model derived from the LCG, we have a state-transition model. Given a state as a conjunction of grounded `shape`, `distX` and `distY` atoms, and an action, we can compute the next state `distX` and `distY` (by first computing `displX` and `displY`) and their probability distributions.

This model also gives an action representation. An action representation consists of a set of rules, each rule a four-tuple: $(action, precond, effects, prob)$. The precondition is a conjunction of the `shape`, `distX` and `distY` atoms for the objects whose relative distances change during the action. The effect is a delete list containing a conjunction of the `distX` and `distY` in the precondition, and an add list containing a conjunction of `distX` and `distY` with the new distance values given by the model (e.g., as in Equation 5.1). The probability is given by the distribution of the DC clauses.

For our running example, the (grounded) action representation of the tap action from S_0 to S_1 is:

$$
\begin{array}{ll}
\textit{action}: & tap(o_r) \\
\textit{precond}: & shape(o_r, cube), shape(o_l, cube), \\
 & distX(o_r, o_l, 7), distY(o_r, o_l, 17) \\
\textit{effects}: & \neg distX(o_r, o_l, 7), \neg distY(o_r, o_l, 17), \\
 & distX(o_r, o_l, 6.99), distY(o_r, o_l, 14.15)
\end{array}
$$

At this point we can use our DCs program and state-transition model. Given S_0 and S_2, we can find the best set of left and right arm actions by forward or backward state-space search over the two time steps, and also in our model we can compute $P(twoArmA(A_L, O_L, A_R, O_R)|S_0, S_2)$ for any action A_L and A_R and objects O_L and O_R.

Note that to restrict the size of each state, and that of the action representation, in this two-step planning problem we can restrict the states to contain only the subset of objects close enough to be manipulated or to interact with objects that are manipulated.

We now have a full two-arm relational affordance model for settings where the two-arm manipulation can be approximated by a combination of the one-arm actions composing it, which we can evaluate in an action prediction setting, which will be Step 3 of our pipeline.

5.5 Evaluation and Results

We want to investigate whether our two-arms probabilistic relational affordance model can be used successfully in a table-top object manipulation setting. This constitutes **Step 3** of our pipeline. We do this in the context of action prediction, where the robot needs to pick the object(s) to act on and the best left and right arm actions to achieve the required effects. We want to find out:

(i) Does our continuous domain model have a higher action prediction rate than the previous discretised model?

(ii) Can the robot pick the correct object(s)?

(iii) Can it pick the correct left and right arm actions?

(iv) Can the robot handle tasks suitable for two-arm manipulation?

We use a table-top setting based on the multiple object action prediction setting introduced in Chapter 4. Objects are placed on the table in front of the robot in two layers as in Section 4.5. Each layer has either one *bar*, or three objects that can be of any of the other three shapes. We generate random shapes for the objects. We ignore the trivial setting with two bars (we want more objects than the number used in the babbling phase), so we generate settings with either four or six objects. Objects in the front layer are always in the field of action of the robot, but not necessarily in the action space of both arms. The objects placed behind might interact with them during an action. All objects are randomly placed within certain margins. Figure 5.5(l) shows one such placement. In each placement we extract O, which is a set containing the values of the shapes of the objects and of the $distX$ and $distY$ for all (ordered) pairs of objects.

We then execute all possible combinations of two-arm actions that meet our two-arm action constraints from Section 5.4.1 with the PR2 (but we ignore the trivial case where both arms tap the same object, where final configuration of objects would be similar to initial). We retain all those where the inverse kinematics succeeds and the arms act on the object(s). We get a dataset containing a set of real-world goal effects E matching the O. E contains a set of $displX$ and $displY$ values for all the objects in the scene. Figure 5.5(r) shows one such goal, which was obtained from the initial setting in Figure 5.5(l) by executing the two sequential actions: left-arm tap on the (left) blue prism (o_1), right-arm push of the red cube (o_2). We use our model to infer the most likely two-arm actions to achieve E, and compare these against the ground truth actions performed when obtaining the dataset.

We generated 100 such settings. One such setting, for Figure 5.5, where we preprocessed the E to transform the object displacements in a goal state consisting of relative object distances is:

Initial: $\texttt{shape}(\texttt{o}_1, \texttt{prism}), \texttt{shape}(\texttt{o}_2, \texttt{cube}), \texttt{shape}(\texttt{o}_3, \texttt{pr}...,$
$\texttt{distX}(\texttt{o}_1, \texttt{o}_2, -1.51), \texttt{distX}(\texttt{o}_1, \texttt{o}_3, -1.37), ...,$
$\texttt{distY}(\texttt{o}_1, \texttt{o}_2, -20.59), ...$
Goal: $\texttt{distX}(\texttt{o}_1, \texttt{o}_2, 5.68), \texttt{distX}(\texttt{o}_1, \texttt{o}_3, -2.21), ...,$
$\texttt{distY}(\texttt{o}_1, \texttt{o}_2, -15.60), ...$

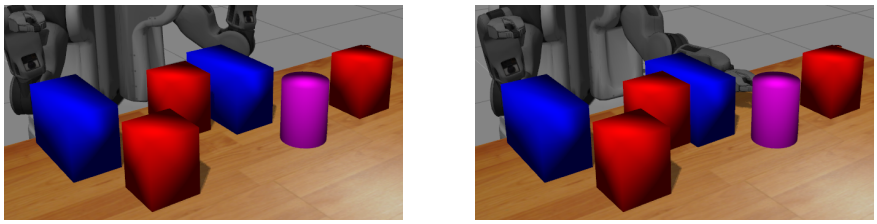


Figure 5.5: Initial object placement (l), and its goal final object locations (r)

We used the PR2 robot in the Gazebo simulator as described in Section 5.3.1. Given that the inverse kinematics trajectory planning has a *2cm* tolerance around the desired goal, our target displacement effects E' in our continuous domain setting are the 4*cm* interval centered on the ground truth dataset values E. We use the SRL model to predict the left and right arm action-object pairs by calculating $\arg\max_{A_L, O_L, A_R, O_R} P(twoArmA(A_L, O_L, A_R, O_R)|O, E')$ and compare these to the ground truth action-object pairs. Table 5.4 summarises the results of the experiments.

Table 5.4: Two-arm model action prediction.

	Total exp.	Success	Pct.
Correct two-arm object(s)/actions	100	68	68%
Correct manipulated objects	100	74	74%
Correct left/right actions	100	68	68%
Random choice			2.78%
Random choice given constraints			9.52%

The robot picks the correct two-arm actions and object(s) to act on in 68% of the cases. For comparison, the original discretised single-arm single-action affordance model presented in Chapter 4 was 58% accurate, and it used a simpler setting with only two shapes. This shows the answer to question (i)

of our evaluation: our continuous extension is better suited for modelling this setting than our previous discretised model.

Furthermore, we predict the correct object(s) to act on in 74% of the cases. The pair of left/right actions are predicted correctly 68% of the time. This shows that our two-arm model can be used by the robot to infer which object(s) and which left/right actions to use to reach a goal in a manipulation setting (questions (ii), (iii) of our evaluation).

We also included in Table 5.4 the random prediction baselines. The probability of randomly picking the correct two-arm object(s)/actions is only 2.78%. This increases to 9.52% if we restrict the actions according to the task constraints introduced in Section 5.4.1.

Qualitatively, many of the random settings in the dataset are good showcases for two-arm manipulation (question (iv)). The four-object settings are either similar to the one in Figure 5.1 if the *bar* is in the back layer, or if it is in the front layer by pushing it with both arms it can interact with two or even all three of the back layer objects which would not be the case with single-arm actions. Settings such as Figure 5.5 illustrate another two-arm scenario: one arm taps an object which by interacting with a second object makes the latter closer for the other arm to act on.

Experiments were run on computers with Intel Core $i5-2500$ $3.3GHz$ processors, $6MB$ cache, and $8GB$ memory. We implemented our model with DCs and used 10000 samples for computing the query probabilities. Each query $P(twoArmA(A_L, O_L, A_R, O_R)|O, E')$, computed as its equivalent form $\frac{P(twoArmA(A_L,O_L,A_R,O_R),E'|O)}{P(E'|O)}$ for better inference performance, took about 30 seconds.

There are several limitations of this approach. Firstly, we have to note that most limitations presented in Chapter 4, including action space, objects interfering with actions, or object orientation, are also present here. Secondly, we have to note that our approach for modelling relational affordances for two-arm robots only works for two-arm actions whose effects are the same, or can be approximated well enough by the effects of the two arm actions composing them. While many two-arm actions fall into this category, some do not, and our approach will not work in those cases. We now present some cases where our approach will have problems. If the first action is a prerequisite for the second action and so the second action cannot be executed on its own, then for this approach to work the robot must try in the babbling stage to execute the second action also in states obtained by the execution of the first action. An example of this case is turning on the TV, then changing the channel. The approach will work only if in the babbling stage the robot tries pressing the channel button

both when the TV is on and off. Another case when the approach will not work is when the effects of two-arm actions are unpredictable from the individual action effects. One example of this is a chemical reaction.

5.6 Conclusion and Future Work

We have presented in this chapter an extension of the relational affordance model for the continuous domain for two-arm robots. We showed that such a model can be used for two-arm manipulation in a multiple object scene, as shown in our experiments on action recognition. Future work will investigate the use of different additional spatial relations and using other background rules for improving action recognition performance, and more complex environments. We also want to investigate sequences of two-arm actions to achieve a task.

Chapter 6

Multiple-Action Two-Arm Manipulation Tasks

This chapter[1] extends the concept of relational affordances in a continuous setting introduced in the previous chapter, in order to be able to tackle more complex manipulation tasks in a table-top environment. We use the two-arm robot relational affordances, as introduced in the previous chapter, to be able to create a plan consisting of several actions that achieve a given goal. In this chapter we will enable the robot to get closer to solving object arrangement tasks in a table-top scenario such as the shopping shelf scenario in Figure 4.2.

6.1 Introduction

Our ultimate goal is for a humanoid robot to be able to achieve a table-top object manipulation and arrangement task, involving multiple manipulation actions on various objects. For this purpose, we will make use of relational affordances to model interactions and spatial relations between multiple objects in the environment, as well as build on continuous and two-arm models. Through interaction with the environment, the robot will learn a relational affordance model for its basic actions. For single-arm actions, the model will be transferred to the other arm in order to obtain a model of two-arm object manipulation.

[1] This chapter is based on work to be submitted to the Robotics and Autonomous Systems journal: Moldovan, B., Nitti, D., Moreno, P., M., Santos-Victor, J., and De Raedt, L., Using Relational Affordances for Multiple-Action Two-Arm Manipulation Tasks.

Once this model is learnt, the robot is given a high-level object arrangement task in terms of relations between objects (e.g., place all the cylinders to the left of the cubes). The task can be achieved by a sequence of actions, each of them involving any of the two hands of the robot. The robot will need to create a plan composed of its basic actions that can achieve the goal. We will illustrate the overall approach with an iCub robot in a table-top setting.

This chapter will build upon the previous affordance model extensions presented in previous chapters in order to tackle a more complex task, that can be solved by planning and executing a sequence of actions. We will use a continuous domain, as well as both arms of the robot, although there are differences in the two-arm modeling with respect to the previous chapter which went more in depth on this topic. We have a bigger repertoire of actions by parameterising the actions, as well as more objects than in the previous experiments. We will focus here on how to use the learnt relational affordance models in order to plan a sequence of actions for a given manipulation task.

6.1.1 Problem Statement and Approach

We will tackle a table-top scenario where a two-arm robot needs to manipulate multiple objects, that can interact with one another, in order to reach a given goal. To reach the goal, the robot will need to execute a sequence of its basic actions, whose relational affordance model it previously learns during a behavioural babbling stage.

One example of such setting can be seen in Figure 6.1. The robot will use its perception to detect the initial setting of the objects, as in Figure 6.1 (left). The robot is then given a high-level goal, specified as spatial relations between the objects in the setting. In this case, the goal is to place the long prisms to the left of the small prisms (as seen from the robot's point of view), while all objects need to be "in the shelf" (considered at the back of the table, behind the dashed line). One possible goal configuration the robot can reach can be seen in Figure 6.1 (right). To achieve this task, the robot first needs to: (1) tap the red object with the right arm (towards the left), then (2) tap the magenta object with the left arm (towards the right), and finally (3) push the magenta object with any of its arms. Note for example that in step (2) the object is only reachable with the left arm, and after it is tapped it is in a position where it can be acted upon with both arms.

This full object arrangement planning task is based on previous relational affordances models presented in Chapter 4 and Chapter 5, extended with parameterised actions, high-level goals, and a planning algorithm, in order to fully model a table-top task for a robot. The general relational affordance models

from the previous chapters defined a joint probability distribution over O, A, E. Here, we go further by defining an action representation which defines states, action preconditions and effects, which is able to handle temporal domains, being related to a probabilistic STRIPS representation as it will be explained in more detail in Section 6.3.

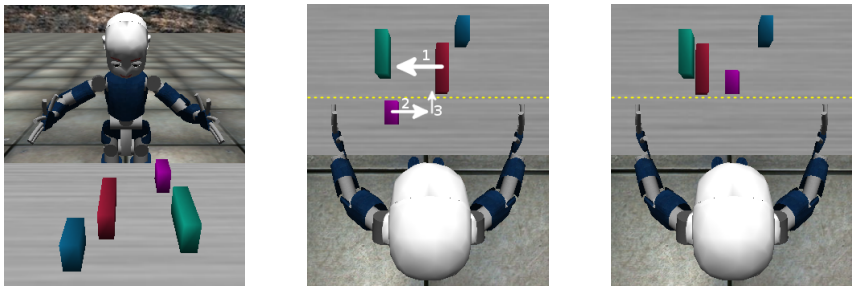


Figure 6.1: Table-top scenario with sequence of arm actions for object placement: (left): initial setting, (middle): actions to reach goal, (right): possible goal arrangement.

Tackling such object manipulation scenario is composed of three different tasks, shown in Figure 6.2:

- Task (i) is learning a two-arm continuous relational affordance model: *given:* a) a set of corresponding O, A, E values collected from exploratory one-arm and simultaneous two-arm action executions in two-object environments, and b) background information about symmetries of left and right arm actions, *find:* c) a continuous setting relational affordance model of two-arm actions, modelled in a PPL.

- Task (ii) is learning a state transition model: *given:* a) the relational affordance model learnt in Task (i), and b) a set of task constraints rules for determining applicable actions (e.g., do not act on potentially occluded objects), *find:* c) a state transition model for the robot actions, to be used for planning tasks.

- Task (iii) is the planning task used to evaluate our model: *given:* a) an initial scene from which using its perception the robot extracts the set of object properties values O, and b) a target goal, given as a set of spatial relations between the objects, together with, c) the state transition model for the robot actions obtained by Task (ii), *find:* d) the next best action to execute towards reaching the goal.

The robot can then repeat the next best action inference and action execution until the goal is reached, or until we reached a predefined maximum number of actions.

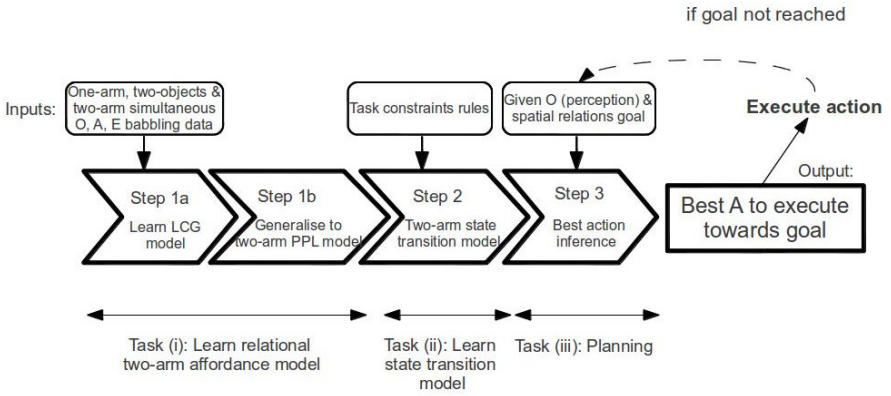


Figure 6.2: Pipeline for table-top two-arm object manipulation.

To solve these tasks, several steps are required, as shown in Figure 6.2:

- 1a) learn a Linear Continuous Gaussian (LCG) Bayesian Network (BN) from single arm and simultaneous two-arm exploratory data,
- 1b) from the LCG model, build the two-arm continuous domain relational affordance model in a PPL,
- 2) build a state transition model from the relational affordance model, and
- 3) infer best action to execute to reach goal (step repeated until goal reached).

Thus, the planner will use the low level information it acquires from its sensors, and is guided by the state transition model obtained from the previously learnt relational affordance model, together with background knowledge about the actions containing high level information about its available set of actions.

6.1.2 Contributions and Outline

The main contribution of this chapter is extending the relational affordance models presented in previous chapters towards a more extensive multiple-actions,

two-arm object manipulation in a table-top setting, including parameterised actions, high-level goals, and a planning algorithm.

In our approach in this chapter, actions will be parameterised and so the robot will be provided with a bigger set of actions. Parameterised actions give a wider skill set, which enlarges the possible number of setups, allowing the robot to perform more tasks than before.

The goal presented to the robot will be a high-level goal composed of spatial relations between the objects. This is in accordance with human-robot interaction approaches and represents a more realistic goal a human would ask from a robot, which has a more compact representation and could be easily explained in natural language. For example, a human would ask the robot to place a cup near the plate to the right of the glass, rather than define a goal based on specific x-y coordinate locations for all the objects in the scene.

Finally, we use the relational affordance model in order to define a state transition model in the table-top setting, and we provide a planning algorithm to be used to infer the next best possible action for the robot to execute towards reaching the given goal. The system will be tested with a real iCub robot with its perception used to detect objects, and with its available motor skills (while the model learning will be done in the iCub simulator).

We continue presenting the basic skills of the iCub robot, which is used for this work, in Section 6.2. Our relational affordance formalism is shown in Section 6.3. In Section 6.4 we describe our approach for learning a relational affordance model from babbling data, and in Section 6.5 we describe our approach for using this model in a planning setting. Finally, Section 6.6 presents experimental results, Section 6.7 presents related work, and we conclude in Section 6.8.

6.2 Basic Skills of the Robot

We employ, both in a real setting and in simulation, the *iCub humanoid robot* [Metta et al., 2008], which has a head with two cameras, two arms and two legs. The legs of the robot are immobile, and we use both arms, and both cameras. Each arm has a force-torque sensor that allows control them in impedance mode from the shoulder joint to the wrist joint. However, the finger joints can be controlled only in position mode. The cameras allow to obtain the disparity map using stereo algorithms.

We assume the robot is provided with a set of core motor actions and perceptual skills. The motor actions are based on a Cartesian controller [Pattacini et al., 2010], which allows to move between points in the space

considering constraints in hand orientation and forces being exerted. The perceptual skills are based on color segmentation in images and stereo vision algorithms that allow to locate objects and their respective sizes. We build on these elements the basic skills of the robot: motor skills to perform the actions and perceptual skills to measure object features and effects.

Motor actions are parameterised with the distance in *cm* and the direction that the hand moves over. The distance is relative to the current location of the objects, while the direction of the actions could be from left to right, right to left and nearer to further. The location of landmark points associated to the direction of the action provides the initial points in space. A point on the left side of the object will be the initial point for left-right direction, the bottom point of an object will be associated to nearer-further direction and so on.

Figure 6.3 illustrates the detected action points on the segmented image of four objects. According to the position of the object's centroid relative to the robot, the right and left landmark points were either retrieved from the image or estimated from the other points. If the centroid was on the left side of the robot, the right point was retrieved and the left one estimated. Similarly, if the center was on the right side, the left point was retrieved and the right one estimated. The bottom points was always retrieved from the image.

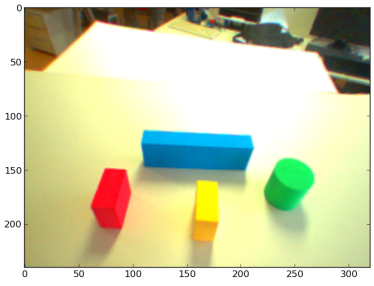
The action execution procedure is provided with a force trigger, which is activated when the magnitude of the force is above a previously defined threshold. The motion of the arms is performed by a minimum-jerk Cartesian controller which reaches a position as close as possible to a given target position while coping with the kinematic and dynamic constraints of the iCub [Pattacini et al., 2010]. Figure 6.4 illustrates the object locations before and after every action execution by the real iCub.

Regarding visual perception, we assume object identification is provided by color segmentation, and object size is provided by stereo vision. Color segmentation is based on an algorithm relying on a synergistic approach combining a confidence-based edge detector and mean shift segmentation [Christoudias et al., 2002]. The image segmentation algorithm is applied on both cameras in order to find the enclosing region of objects on each image. Then, the centroid of the segmented region is extracted on both cameras in order to perform stereo triangulation. This process provides the 3D position of the object's centroid, which represents its location.

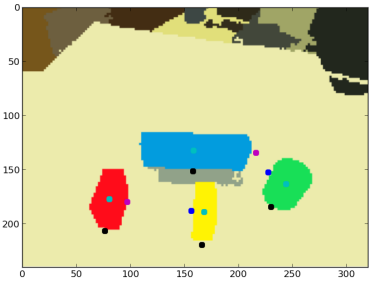
Object size is computed from the combination of segmentation and stereo algorithms. From the segmented image of one of the eyes, we extract the ellipse that encloses each coloured region[2]. The points in the ellipse that intersect its

[2]having the same normalized second central moments

(a) Robot and objects



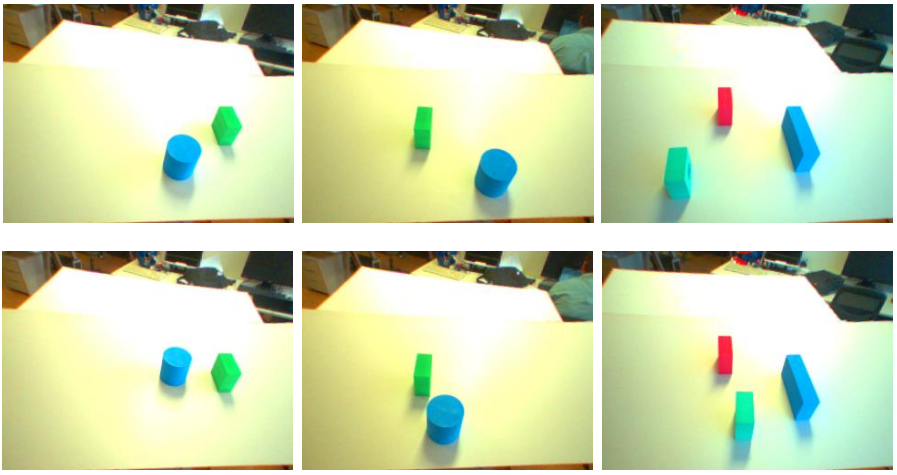
(b) Raw image



(c) Segmented image with landmark points

Figure 6.3: Illustration of the table-top scenario for the real iCub, with its correspondent point of view of the robot and the segmentation result. The coloured points are associated to points as follows: cyan to centroids, black to bottom, magenta to right and blue to left.

major axis are mapped onto the stereo disparity image for 3D perception. The distance between the 3D mapped ellipse points provide the object size. Figure 6.5 illustrates the object size computation.



(a) Push (Left arm, 15cm) (b) Tap (Right arm, 10cm) (c) Tap (Left arm, 20 cm)

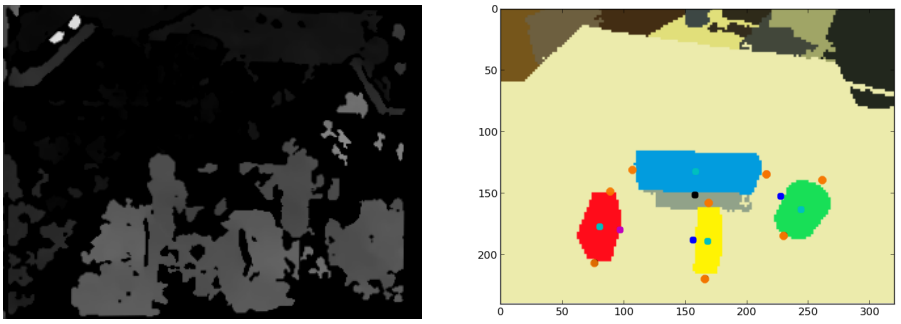
Figure 6.4: Action execution examples from the iCub's left camera point of view. The top row images show the location of the objects before the action execution and the bottom row images show their locations after action execution. Each column represent a different action

6.3 Relational Affordances in a Planning Setting

We will now introduce a formalism that can be used to describe our concept of relational affordances for solving table-top planning tasks. This is an extension of the more general relational affordance formalism introduced in Section 3.4. This general relational affordance formalism defined a joint probability distribution over O, A, E. Here, in order to be able to tackle a temporal domain and a planning task, we will additionally define a state transition model and action representation.

We will follow the notation from Section 3.4, which defined relational affordances. In addition, for convenience, we will use here for object properties random variable atoms the shorthand notation $\mathtt{op_i}(\mathtt{Z})$, and for the effects random variable atoms $\mathtt{e_i}(\mathtt{Z})$. In our setting, the arity of these atoms is either one or two, but the approach can be generalised.

Since our planning task involves multiple time steps, we need to deal with a temporal domain. Therefore, for this purpose, as opposed to Section 3.4 which used DCs notation, we will employ a DDCs-style syntax next. As a reminder, in DDCs, a t or $t+1$ subscript for an atom means that the atom is defined for the state at time t or $t+1$ respectively. We will now define a state description



(a) Disparity image (b) Segmented image with landmark points

Figure 6.5: Illustration of the object size computation. Left-hand image shows the disparity map of the example shown in Figure 6.3. The orange points in the right-hand image show the points that intersect with the ellipse's major axis. The orange points are mapped onto 3D using their associated disparity value, and the 3D distance between each pair is defined as the object size.

and an action representation based on our affordance model.

6.3.1 State description

A state represents a description of the environment of the robot at a given time. A state is a conjunction of all grounded atoms $\mathtt{op_i(Z)}$ for all the objects in the domain $\mathcal{Z}$.

Example 6.1. *In the table-top setting from Example 3.1 from Section 3.4, considering just the object properties* `color` *and* `distance`*, we could have the following state at time t:*

$$\begin{aligned}&\mathtt{shape(ball)_t = sphere}, \mathtt{shape(book)_t = prism},\\ &\quad \mathtt{shape(cup)_t = cylinder}, \mathtt{distance(ball, book)_t = 3}\\ &\quad \mathtt{distance(ball, cup)_t = 8}, \mathtt{distance(book, cup)_t = 2}.\end{aligned}$$

6.3.2 Action representation

We define action representation as a set of rules. We will follow the probabilistic-STRIPS definition of a rule as a four-tuple: $< a(Z), pre, \mathit{eff}, prob >$, but as mentioned previously, we use DDCs-style syntax. Applying an action $a(Z)$ in

the current state S_t at time t will result in the next state S_{t+1} at time $t+1$ with probability *prob*. This *rule probability* is given by $prob = P(S_{t+1}|a(Z), S_t)$.

The *preconditions pre* are a set of relations on the random variable atoms $\mathtt{op_i(Z)}$ from O. An action $a(Z)$ can be executed in the current state S_t only if there is a substitution θ for the variables in *pre* such that $pre\theta$ holds in S_t. If $pre\theta$ holds, we say that the action is applicable.

Example 6.2. *Consider a setting with current state S_t shown in Example 6.1.*

For the action $\mathtt{push(Z_1)}$ *the preconditions can be:*

$$\begin{aligned}\mathtt{pre(push(Z_1)_t)} \leftarrow &\simeq(\mathtt{shape(Z_1)_t}) = \mathtt{sphere}, \mathtt{object(Z_2)}, \mathtt{Z_2} \neq \mathtt{Z_1},\\ &\simeq(\mathtt{distance(Z_1, Z_2)_t}) < 5.\end{aligned}$$

where $\mathtt{object(Z_2)}$ *holds if* $\mathtt{Z_2}$ *is an object in the scene. The preconditions hold for the substitution:* $\theta = \{Z_1/ball, Z_2/book\}$.

The *effect eff* describes the state change after the execution of the action $a(Z)$ and consists of an add and a delete list: $< add, del >$. In DDCs, this can be achieved by defining a state transition model. So, for each atom in the add list, that will be defined in the next state at time $t+1$, we need to define a state transition model clause as in Section 2.3.3: $\mathtt{h_{t+1}} \sim \mathcal{D} \leftarrow \mathtt{body_t}$.

Note that for DDCs, if a random variable is not defined for state $t+1$, it remains undefined, this in fact constituting an atom of the delete list.

Compared to the rule effects eff, in our table-top setting affordance effects E represent relative changes in one or more object properties O due to the action. The affordance effects can be seen as a delta change between the states S_t and S_{t+1}. Therefore, in the simplest case, if the precondition of an action holds and the action is executed, the value of an object property $\mathtt{op_i(Z)}$ in the next state can be obtained by adding the effect $\mathtt{e_i(Z)}$ of the action to its value in the current state:

$$\begin{aligned}\mathtt{op_i(Z)_{t+1}} \sim \mathtt{finite}([\mathtt{prob} : \mathtt{NewVal}]) \leftarrow &\mathtt{pre(a(X)_t)}, \mathtt{a(X)_t},\\ &\mathtt{NewVal} = \simeq(\mathtt{op_i(Z)_t}) + \simeq(\mathtt{e_i(Z)_t}).\end{aligned}$$

where `prob` is the rule probability value. Of course, in more complex cases, obtaining the next state can require more complex rules involving more than one object property and effect. For example, the state can be defined by the x and y-coordinates in 2D space, and the effects can be the straight-line displacement distance and angle of orientation, in which case computing the next state involves taking into account both effects in the same rule.

Example 6.3. *Let us consider the setting in Example 6.2, extended to include the following affordance effects for that* $\mathtt{push(Z_1)}$ *action:*

Effects: $\texttt{rel\_displacement}(\texttt{Z}_1, \texttt{Z}_2)_\texttt{t} = -2$

For substitution: $\theta = \{Z_1/ball, Z_2/book\}$*, the action is:* $\texttt{push}(\texttt{ball})_\texttt{t}$*, and the affordance effects:* $\texttt{rel\_displacement}(\texttt{ball}, \texttt{book})_\texttt{t} = -2$.

The state transition model clause for the `distance` *object property could be:*

$$\begin{aligned}&\texttt{distance}(\texttt{Z}_1, \texttt{Z}_2)_{\texttt{t}+1} \sim \texttt{finite}([0.8 : \texttt{NewVal}, 0.2 : \texttt{OldVal}]) \leftarrow \\ &\quad \simeq(\texttt{shape}(\texttt{Z}_1)_\texttt{t}) = \texttt{sphere}, \simeq(\texttt{distance}(\texttt{Z}_1, \texttt{Z}_2)_\texttt{t}) < 5, \texttt{push}(\texttt{Z}_1)_\texttt{t}, \\ &\quad \texttt{OldVal} = \simeq(\texttt{distance}(\texttt{Z}_1, \texttt{Z}_2)_\texttt{t}), \\ &\quad \texttt{NewVal} = \texttt{OldVal} + \simeq(\texttt{rel\_displacement}(\texttt{Z}_1, \texttt{Z}_2)_\texttt{t}).\end{aligned}$$

Given the state at time t *from Example 6.1, and assuming an existing state transition clause that maintains the shape of the objects, then at time* $t+1$ *with a probability of* 0.8 *the next state will be:*

$$\begin{aligned}&\texttt{shape}(\texttt{ball})_{\texttt{t}+1} = \texttt{sphere}, \texttt{shape}(\texttt{book})_{\texttt{t}+1} = \texttt{prism}, \\ &\quad \texttt{shape}(\texttt{cup})_{\texttt{t}+1} = \texttt{cylinder}, \texttt{distance}(\texttt{ball}, \texttt{book})_{\texttt{t}+1} = 1, \\ &\quad \texttt{distance}(\texttt{ball}, \texttt{cup})_{\texttt{t}+1} = 8, \texttt{distance}(\texttt{book}, \texttt{cup})_{\texttt{t}+1} = 2.\end{aligned}$$

6.3.3 Comparison to General Model

The general relational affordance model introduced in Section 3.4 defined affordances as a joint probability distribution over O, A, E. In this chapter, we build on top of that definition, in order to build a relational affordance model that can handle a temporal domain and a planning task. For this purpose, we define an action model, including action preconditions based on the affordance object properties, and a state transition model which defines the next state based on the object properties and affordance effects of the current state. This relational affordance model is related to probabilistic STRIPS, and is now able to handle the table-top planning task of this chapter.

We note that, unlike this relational affordance model to be used for planning tasks, the general relational affordance model from Section 3.4 also defines the probability distribution over actions given a set of object properties. This means that to obtain a general affordance model from the model used for planning in this chapter, one would need to define a policy, which would define the probability distribution over actions given a state as a conjunction of grounded object property atoms. Note that the naive planning algorithm we will propose in Section 6.5.3 samples the next state following a uniform policy over the applicable actions in the current state (i.e., the actions whose preconditions hold in the current state).

For example, assume an action *pickup* can be executed on an object if two

preconditions hold: the object is less than $10cm$ away and the object type is a cup. Now imagine some very noisy sensors, which sometimes tell us that the object type is a cup, even if in reality it is a ball, and that have bigger errors in measuring distance. We want to model a policy that a *pickup* action should be executed only in 80% of the cases when our sensors detect a cup less than $10cm$ away, while in the other 20% of the cases the robot should move closer to the object to reduce uncertainty. In this case, using the general affordance model defined as the joint probability distribution $P(O, A, E)$ is beneficial.

Another type of problems the more general joint probability distribution approach solves can be seen in the next chapter when performing object search. In that particular case, the features available (e.g., width, height) cannot fully determine the action afforded by an object (e.g., pouring), but only a probability distribution over actions.

6.4 Learning the Affordance Model

We will now present the learning of the relational model, in a continuous domain, which constitutes Task (i) of our problem, and which is composed of **Step 1a** and **Step 1b**.

The **object properties** O are the same as in Chapter 5: *shape*, and *relations* relative distance along the x-axis ($distX$) and y-axis ($distY$) between two objects (in cm). As opposed to Chapter 5, the centroids of the objects (from where the distances are measured) and determined from perception by using the object segmentation algorithm. These object properties $distX$ and $distY$ are shown in Figure 6.6(l) with an iCub robot in simulation, with the objects' position before (l) and after (r) an action (tap) execution. The x and y-axes correspond with the iCub x and y-axes from the robot's viewpoint.

In this setting we use a different set of objects than modelled before. We will use five different objects of one of three different shapes. The shape of two of the objects is small prism ($sprism$) (sizes: $4cm \times 8cm \times 8cm$ and $4cm \times 9cm \times 7.5cm$). There are two big prisms ($bprism$) (sizes: $4cm \times 14.5cm \times 9cm$ and $4cm \times 16cm \times 8cm$). There is one long bar ($lbar$) (size: $27cm \times 6cm \times 5cm$).

As before, the **action** A is one of two basic arm core motor actions: *tap* (right-to-left hand movement for the right arm, left-to-right for the left) and *push* (away movement for both arms). But as an extension of the previously presented affordance models, the action is parameterised by the distance in cm that the arm is moved from its pre-action position next to the object until the action is finished. The arm movement distance values can be one of: $10cm$ and $20cm$

for the *tap* actions, and 15*cm* and 25*cm* for the *push* actions (but more can be modelled as well). Similar to previous chapters, these values were chosen considering the sizes of the objects involved.

We will also have a two-arm simultaneous *push* action on the same object. Alternatively, this could also have been modelled by a combination of single-arm actions as in the previous chapter, but because of iCub limitations and to increase accuracy, the robot will directly learn from two-arm push babbling data.

We will use the same **effects** E as before. We use the displacements along the x-axis ($displX$) and y-axis ($displY$) of the centroid of each object. These effects $displX$ and $displY$ are shown in Figure 6.6(r), which overlays the initial objects' positions over their final positions.

To learn an affordance model, the robot first performs a behavioural babbling stage, in which it explores the effect of its actions on the environment. For this behavioural babbling stage, for the single-arm actions the robot uses its right-arm only. For these actions a model of the left-arm will be later built by exploiting symmetry as in Section 5.4. We include the simultaneous two-arm *push* on the same object in the babbling phase, allowing for a more accurate modelling of action effects for the iCub. As opposed to the two-arm affordance modelling from the previous chapter, we also include in the babbling phase the two-arm simultaneous actions whose effects might not always be well modelled by the sum of the individual single-arm actions.

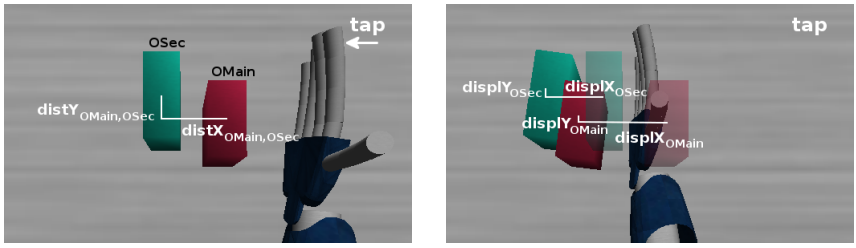


Figure 6.6: Relational O before (l), and E after the action execution (r).

The babbling phase consists of placing pairs of objects in front of the robot at various positions, and executing one of its actions A described above on one object as described in Section 5.3. For choosing these babbling settings we use a similar approach as that introduced in Chapter 4, and later used in Chapter 5. We place one object in the centre of the action space for the respective action, and we place the second object at varying x and y-coordinates around this first object such that the distance between the objects is smaller than the hand motion distance.

The robot executed 200 such exploratory actions. One example of collected data during such an action is shown in Table 6.1, for the right-arm action ($tap(10)$) execution in Figure 6.6.

Table 6.1: Example collected O, A, E data for action in Figure 6.6

Object Properties	**Action**	**Effects**
$shape_{O_{Main}} : sprism$ $shape_{O_{Sec}} : sprism$ $distX_{O_{Main},O_{Sec}} : 6.94cm$ $distY_{O_{Main},O_{Sec}} : 1.90cm$	$tap(10)$	$displX_{O_{Main}} : 10.33cm$ $displY_{O_{Main}} : -0.68cm$ $displX_{O_{Sec}} : 7.43cm$ $displY_{O_{Sec}} : -1.31cm$

Note that these values are obtained by the robot from its perception, which naturally introduces uncertainty, which the relational affordance model takes into account (e.g., the displacement of O_{Main} is observed to be a bit more than $10cm$).

As in Chapter 5, we will first learn the LCG BN from this data, which is **Step 1a** of our approach. This LCG BN has the same structure as the one shown in Figure 4.8 in the previous chapter. We then learn the parameters of this LCG BN. For example, during our *tap(10)* action for two interacting cubes (as in Figure 6.6), the displacement of O_{Sec} on the x-axis is (in cm):

$$\mathcal{N}(7.05 + 0.57 * distX_{O_{Main},O_{Sec}} + 0.02 * distY_{O_{Main},O_{Sec}}, 0.41). \tag{6.1}$$

This makes sense intuitively as the second cube is moved along by the $tap(10)$ action, so we expect its displacement to depend mainly on $distX$, but also a little bit on $distY$ if the objects are not aligned, as in Figure 6.6.

Similar to Chapter 5, we will model this LCG BN using DCs syntax, which is **Step 1b** of our approach.

For the modelling of affordances in PPL for the task in this chapter, the main predicates we will use are presented in Table 6.2 below.

For example, to transform the LCG Equation 6.1 in DCs, one writes:

```
displX(ObjSec) ~ gaussian(Mu, 0.41) ← tap(ObjMain, 10),
    shape(ObjMain, sprism), shape(ObjSec, sprism),
    Mu is 7.05 + 0.57* ≃(distX(ObjMain, ObjSec))+
    0.02* ≃(distY(ObjMain, ObjSec)).
```

meaning for a tap action, if the two shapes are small prisms, `displX` of `ObjSec` is distributed according to a Gaussian with mean given by Mu, as in Equation 6.1.

Table 6.2: Predicates used for affordance modelling

Predicate	Meaning
shape(Obj, Shape)	The shape of object Obj is Shape.
distX(Obj1, Obj2)	Distribution of the relative x-axis distance between objects Obj1 and Obj2.
distY(Obj1, Obj2)	Distribution of the relative x-axis distance between objects Obj1 and Obj2.
displX(Obj)	Distribution of the x-axis displacement of Obj.
displY(Obj)	Distribution of the y-axis displacement of Obj.
tap(Obj, Arm, Param)	A parameterised tap on object Obj where the arm Arm is moved a distance of Param. Arm can be left or right.
push(Obj, Arm, Param)	A parameterised push on object Obj where the arm Arm is moved a distance of Param.
push2hand(Obj, both, Param)	A parameterised push on object Obj where the both arms are moved a distance of Param.
approx_ok(A, ObjM, ObjS, DX, DY)	True if the Gaussian approximation for the action effect holds for the action A on main object ObjM and with secondary object ObjS when the x-axis distance between the objects is DX and the y-axis distance between the objects is DY.
coordX(Obj)	Distribution of the x-axis coordinate of object Obj.
coordY(Obj)	Distribution of the y-axis coordinate of object Obj.
occluded(Obj)	True if object Obj is potentially occluded behind any other object in the scene.
left(Obj1, Obj2)	True if object Obj1 is to the left of object Obj2.
right(Obj1, Obj2)	True if object Obj1 is to the right of object Obj2.
inshelf(Obj1)	True if the y-axis coordinate of the centre of object Obj1 is greater than $40cm$.
outshelf(Obj1)	True if the y-axis coordinate of the centre of object Obj1 is less than or equal to $40cm$.
near(Obj1, Obj2)	True if the centre-to-centre distance between objects Obj1 and Obj2 is less than $10cm$.

Similar to Chapter 5, we need to model that the above Gaussian approximation as in Equation 6.1 holds only in cases when the distance DX and DY between ObjMain and ObjSec is sufficiently small for the action to have an effect on ObjSec. For this we can define the predicate approx_ok, which is true when this is the case.

For example, in our iCub setting, for a *tap*(10) for two small prisms as in Figure 6.6, we can use the definite clause:

$$\texttt{approx\_ok}(\texttt{tap}, 10, \texttt{sprism}, \texttt{sprism}, \texttt{DX}, \texttt{DY}) \leftarrow \texttt{DX} > 4, \texttt{DX} < 14, \\ \texttt{DY} > -8, \texttt{DY} < 8.$$

The smallest x-axis centre-to-centre distance between two objects is $4cm$, so for the objects to interact during a tap their centres need to be between $4cm$ and $14cm$ away on the x-axis. On the y-axis, since the y-axis dimension of a *sprism* is $8cm$, their centres need to be between $-8cm$ and $8cm$ away for an interaction to occur.

Then we just need to add $\texttt{approx\_ok}(\texttt{tap}, 10, \texttt{sprism}, \texttt{sprism}, \texttt{DX}, \texttt{DY})$ to the body of the DC clause defining `displX` above. Doing so the displacement $\texttt{displX}(\texttt{ObjSec})$ will be defined only when `ObjSec` is close to `ObjMain`. Similar rules can be added to enforce the action space.

At this point we can fully model the right arm relational affordance model, as well as the two-arm simultaneous *push*, with the learnt parameters.

For the left arm, given the symmetry of the iCub, the model is equivalent to the model for the right arm mirrored through the plane perpendicular to the table that passes through the centre of the robot. For our iCub model, in the left-arm model all the y-axis values are the same as for the right-arm model, but the x-axis values are the negative of their right-arm equivalent.

In our DC framework, the *displX* and *displY* random variables for left and right arm actions need to be defined by different probability distributions. So, we need to add an extra term to our action atoms to signify the arm performing the action. At this point, we can automatically generate the PPL code for the left arm. For our running example, the equivalent code for Equation 6.1 for the left arm:

$$\texttt{displX}(\texttt{ObjSec}) \sim \texttt{gaussian}(\texttt{Mu}, 0.41) \leftarrow \texttt{tap}(\texttt{ObjMain}, \texttt{left}, 10), \\ \texttt{shape}(\texttt{ObjMain}, \texttt{sprism}), \texttt{shape}(\texttt{ObjSec}, \texttt{sprism}), \\ \texttt{Mu is } -7.05 - 0.57 * \simeq(\texttt{distX}(\texttt{ObjMain}, \texttt{ObjSec})) - \\ 0.02 * \simeq(\texttt{distY}(\texttt{ObjMain}, \texttt{ObjSec})).$$

At this point we have a relational affordance model for single right and left-arm actions, as well as the simultaneous two-arm actions learnt with the babbling data for the iCub. Once the program is defined, the inference algorithm based on sampling from [Gutmann et al., 2011b] or [Nitti et al., 2013] can be used to compute the probability of a query.

For example, given our example, one can ask for the probability of the x-axis displacement of the secondary object o_2 being greater than $5cm$ given some

initial distances between the main object o_1 and o_2:

$P(displX(o_2) > 5 | action(o_1, tap, 10), distX(o_1, o_2) = 7, distY(o_1, o_2) = 2)$.

6.5 Planning

After Task (i) is achieved, the robot has a relational affordance model for modelling single-arm left and right-arm actions, and simultaneous (*push*) two-arm actions. In order for the robot to reach a more complex goal, which requires several actions on objects, we present Task (ii), namely the planning task where the state transition model is obtained from the learnt relational affordance model, accomplished by **Step 2** of our approach. The overall planning model can be seen in Appendix B.

6.5.1 States and Action Representation

To model the state transition model of the dynamics of the world we will use the DDCs formalism. In our table-top object placement setting, states will have to describe the shape and spatial configurations of the objects. The position of an object is defined by the x ($coordX$) and y-coordinates ($coordY$) of its centroid. As opposed to models introduced in previous chapters, we will define a state as a conjunction of the $coordX$ and $coordY$ grounded atoms for all the table-top objects: $\wedge_n coordX(o_i, x_i)_t \wedge_n coordY(o_i, y_i)_t$.

The state before executing an action is obtained by the iCub by using its perception to obtain object shapes and centroids. For example, the initial state S_0 corresponding to Figure 6.6 (l) before the tap can be seen in Table 6.3, where o_r is the right (red) object, and o_l the left one.

Table 6.3: Example states for tap action in Figure 6.6

S_0	S_1
$coordX(o_r)_0 : -10.11$	$coordX(o_r)_1 : 0.22$
$coordY(o_r)_0 : 42.70$	$coordY(o_r)_1 : 42.02$
$coordX(o_l)_0 : -3.17$	$coordX(o_l)_1 : 4.26$
$coordY(o_l)_0 : 44.60$	$coordY(o_l)_1 : 43.29$

As opposed to the previous chapters, we will use high-level goals, represented by a conjunction of spatial relations between objects. For example, the goal can be to place a small prism to the $left$ of a big prism:

$$\texttt{goal} \leftarrow \texttt{shape}(\texttt{O}_1, \texttt{sprism})_\texttt{t}, \texttt{shape}(\texttt{O}_2, \texttt{bprism})_\texttt{t}, \texttt{left}(\texttt{O}_1, \texttt{O}_2)_\texttt{t}.$$

Note that the goal is true if there exists at least a pair of objects O_1 and O_2 that satisfy *goal*. If we are interested to achieve the goal for specific objects, we just need to replace the logical variables O_1 and O_2 with the constants of the objects.

Each spatial relation will be defined in terms of the x and y-coordinates of objects. We will define the following spatial relations for our tasks:

$$\begin{aligned}
&\texttt{left}(\texttt{O}_1, \texttt{O}_2)_\texttt{t} \leftarrow \texttt{O}_1 \neq \texttt{O}_2, \simeq(\texttt{coordX}(\texttt{O}_1)_\texttt{t}) > \simeq(\texttt{coordX}(\texttt{O}_2)_\texttt{t}).\\
&\texttt{right}(\texttt{O}_1, \texttt{O}_2)_\texttt{t} \leftarrow \texttt{O}_1 \neq \texttt{O}_2, \simeq(\texttt{coordX}(\texttt{O}_1)_\texttt{t}) \leq \simeq(\texttt{coordX}(\texttt{O}_2)_\texttt{t}).\\
&\texttt{inshelf}(\texttt{O}_1)_\texttt{t} \leftarrow \simeq(\texttt{coordY}(\texttt{O}_1)_\texttt{t}) > 40.\\
&\texttt{outshelf}(\texttt{O}_1)_\texttt{t} \leftarrow \simeq(\texttt{coordY}(\texttt{O}_1)_\texttt{t}) \leq 40.\\
&\texttt{near}(\texttt{O}_1, \texttt{O}_2)_\texttt{t} \leftarrow \texttt{O}_1 \neq \texttt{O}_2, (\simeq(\texttt{coordX}(\texttt{O}_1)_\texttt{t}) - \simeq(\texttt{coordX}(\texttt{O}_2)_\texttt{t}))^2 +\\
&\quad (\simeq(\texttt{coordY}(\texttt{O}_1)_\texttt{t}) - \simeq(\texttt{coordY}(\texttt{O}_2)_\texttt{t}))^2 < 100.
\end{aligned}$$

The `near` spatial relation is defined as the centre-to-centre distance between the objects being less than $10cm$. The `inshelf` is defined as the y-axis coordinate of the object being greater than $40cm$, corresponding to objects behind the dotted yellow line in Figure 6.1). Similarly, `outshelf` is defined as the y-axis coordinate of the object being less than or equal to $40cm$.

The state transition model can be defined from our learnt affordance model. Using the affordance model, we can compute the object displacements $displX$ and $displY$ caused by an action with its respective probability distribution. More specifically, the new x and y-axis coordinates of an object defining state S_{t+1} at time $t+1$ are given by the sum between the old coordinates in state S_t at time t and the displacement of the object due to the action, or are the same as in the previous state if there is no object displacement. For example, for the x-axis:

$$\begin{aligned}
&\texttt{coordX}(\texttt{O}_1)_{\texttt{t}+1} \sim \texttt{val}(\texttt{X}) \leftarrow \texttt{X is} \simeq(\texttt{coordX}(\texttt{O}_1)_\texttt{t}) + \simeq(\texttt{displX}(\texttt{O}_1)_\texttt{t}).\\
&\texttt{coordX}(\texttt{O}_1)_{\texttt{t}+1} \sim \texttt{val}(\texttt{X}) \leftarrow \texttt{X} = \simeq(\texttt{coordX}(\texttt{O}_1)_\texttt{t}), \texttt{not}(\texttt{displX}(\texttt{O}_1)_\texttt{t}).
\end{aligned}$$

The distribution `val(v)` assigns probability 1 to the value `v`, in this case the value is defined in the body of the clause. While `not(expr)` succeeds if the query `expr` fails or is undefined, in this case when $\texttt{displX}(\texttt{O}_1)_\texttt{t}$ is undefined. In our running example from Figure 6.6 the next state x and y-axis coordinates due to the $tap(right, 10)$ action can be seen in Table 6.3. For example, the new x-axis coordinate for the o_r object in the next state S_1 is: $-10.11 + 10.33 = 0.22$.

6.5.2 Adding Task Constraints in the Environment

The DDC framework allows also for the definition of action applicability. The planning algorithm will only select from the applicable actions, i.e., $\texttt{applicable}(\texttt{action})_\texttt{t}$ holds. In a normal household environment there are many objects present, however acting on all of them might not make sense in order to achieve tasks (e.g., picking up a fork and a book at the same time), or an action might not be possible in a given object configuration (e.g., an object might be hidden behind another object or out of reach). So we want to define $\texttt{applicable}(\texttt{action})_\texttt{t}$ to enforce any task constraints in the environment, which will also narrow down the state space search of the robot when doing planning.

In our approach, we define applicable actions as actions on objects whose centre falls in the robot's action space for the respective action. For example, for the left arm, the applicable action rules are the following:

$$
\begin{aligned}
&\texttt{applicable}(\texttt{tap}(\texttt{Obj},\texttt{left},10))_\texttt{t} \leftarrow \simeq(\texttt{coordX}(\texttt{Obj})_\texttt{t}) > -15,\\
&\quad \simeq(\texttt{coordX}(\texttt{Obj})_\texttt{t}) < 20, \simeq(\texttt{coordY}(\texttt{Obj})_\texttt{t}) > 2,\\
&\quad \simeq(\texttt{coordY}(\texttt{Obj})_\texttt{t}) < 50.\\
&\texttt{applicable}(\texttt{tap}(\texttt{Obj},\texttt{left},20))_\texttt{t} \leftarrow \simeq(\texttt{coordX}(\texttt{Obj})_\texttt{t}) > -15,\\
&\quad \simeq(\texttt{coordX}(\texttt{Obj})_\texttt{t}) < 20, \simeq(\texttt{coordY}(\texttt{Obj})_\texttt{t}) > 2,\\
&\quad \simeq(\texttt{coordY}(\texttt{Obj})_\texttt{t}) < 50.\\
&\texttt{applicable}(\texttt{push}(\texttt{Obj},\texttt{left},15))_\texttt{t} \leftarrow \simeq(\texttt{coordX}(\texttt{Obj})_\texttt{t}) > -10,\\
&\quad \simeq(\texttt{coordX}(\texttt{Obj})_\texttt{t}) < 20, \simeq(\texttt{coordY}(\texttt{Obj})_\texttt{t}) > 25,\\
&\quad \simeq(\texttt{coordY}(\texttt{Obj})_\texttt{t}) < 50.\\
&\texttt{applicable}(\texttt{push}(\texttt{Obj},\texttt{left},25))_\texttt{t} \leftarrow \simeq(\texttt{coordX}(\texttt{Obj})_\texttt{t}) > -10,\\
&\quad \simeq(\texttt{coordX}(\texttt{Obj})_\texttt{t}) < 20, \simeq(\texttt{coordY}(\texttt{Obj})_\texttt{t}) > 25,\\
&\quad \simeq(\texttt{coordY}(\texttt{Obj})_\texttt{t}) < 50.
\end{aligned}
$$

We showed in Section 5.5 how task constraints for two-arm actions were added. Such task constraints can also be modelled with the help of DDCs. As a showcase, we show below the precondition that an action should not be executed on a possibly occluded object, that is an object that is behind another object. We define a possibly occluded object, as any object that has a y-coordinate greater than that of another object placed in front of it, and an x-coordinate of its center within $\pm 10cm$ of that object in front of it. Although this precondition was not used for the evaluation of our approach, it could have been defined for example with the help of logical rules as follows:

$$
\begin{aligned}
&\texttt{occluded}(\texttt{Obj})_\texttt{t} \leftarrow \simeq(\texttt{coordY}(\texttt{SecO})_\texttt{t}) > \simeq(\texttt{coordY}(\texttt{Obj})_\texttt{t}),\\
&\quad \simeq(\texttt{coordX}(\texttt{SecO})_\texttt{t}) > \simeq(\texttt{coordX}(\texttt{Obj})_\texttt{t}) - 10,\\
&\quad \simeq(\texttt{coordX}(\texttt{SecO})_\texttt{t}) < \simeq(\texttt{coordX}(\texttt{Obj})_\texttt{t}) + 10.\\
&\texttt{applicable}(\texttt{push}(\texttt{X},\_,\_))_\texttt{t} \leftarrow \texttt{object}(\texttt{X}), \texttt{not}(\texttt{occluded}(\texttt{X})).
\end{aligned}
$$

Further preconditions for object applicability can be coded depending on the background knowledge one has about the task the robot needs to execute. For example, some actions might only be desirable on certain objects. The DDC makes it easy to define these task constraints with the help of logical rules, which in turn limits the search space of the robot when doing planning.

6.5.3 Proposed Planning Algorithm

Once we have defined the DDC state transition model, we adopt a simple sample-based planner, with the algorithm shown in Algorithm 1. This will allow us to infer the best action we need to execute in order to reach the goal, which is **Step 3** of our approach. This step will be repeated until the goal is reached, or until we reached a predefined maximum number of actions. The algorithm starts with an initial state S_0, a goal G and a reward function $r(S, a|G)$ derived from G, for state S and action a, which we will define shortly. For each applicable action a_0 it samples N_e episodes following a default policy π_D and average the obtained rewards to estimate $Q(S_0, a_0)$. Finally, choose the action that maximize $\arg\max_{a_0} Q(S_0, a_0)$.

Algorithm 1 Planning algorithm for finding the best action towards the goal

1: **procedure** FINDBESTACTION(S_0)
2: **for** $a_0 \in$ {applicable actions in S_0} **do**
3: sample N_e episodes of length d from (S_0, a_0) with default policy π_D
4: $Q(S_0, a_0) \leftarrow \frac{\sum_{t=0}^{d} r_t}{N_e}$
5: **end for**
6: return $\arg\max_{a_0} Q(S_0, a_0)$
7: **end procedure**

The default policy π_D is a uniform distribution in the applicable actions in the state S: $\pi_D(a|S) = uniform(\{\text{applicable actions in } S\})$. In DDC this becomes:

$$\texttt{policy}_\texttt{t} \sim \texttt{uniform}(\texttt{List}) \leftarrow \texttt{findall}(\texttt{A}, \texttt{applicable}(\texttt{A})_\texttt{t}, \texttt{List}).$$

This algorithm is naive because it evaluates the Q function using a flat policy, indeed it does not have a policy improvement mechanism. In a discrete state space policy improvement would be easy to implement, however in a hybrid relational domain this requires a non-trivial representation to store the Q function.

Despite its simplicity, a similar strategy has been used in Monte Carlo Tree Search with the name “Flat Monte Carlo" [Browne et al., 2012] with good

results in computer games. In addition, this algorithm is faster than sparse sampling as showed in preliminary experiments.

In our setting we want to reach a goal G in the minimum number of steps. That is minimizing the expected cost of reaching the goal assuming each action has the same cost c in any state. This is equivalent to maximising the expected reward in an MDP with a reward function $r(S, a|G) = 0$ for $S \models G$, and $r(S, a|G) = -c$ otherwise, with $c > 0$. The value assigned to c is not important (e.g. $c = 1$), since it gives the same policy. In addition, we assume a maximum number of steps d to reach the goal (finite horizon MDP). While the state transition model $p(S_{t+1}|S_t, a_t)$ is defined as a set of DDCs of the form $\mathtt{h_{t+1}} \sim \mathcal{D} \leftarrow \mathtt{body_t}$. $\mathtt{h_{t+1}}$ defines a random variable in the next state S_{t+1} with distribution $\mathcal{D}$ when $\mathtt{body_t}$ holds. The body of the clause includes the action and other conditions that specify when the clause applies.

To implement the planner we use DCPF [Nitti et al., 2014]: a particle filter for DDCs. We initialize the particles with the initial state S_0 and generate episodes with the default policy as described in the algorithm. We assume full observability, thus the observation model is not needed (as in HMMs/DBNs). Therefore, we exploit DCPF only to generate sequential samples and store the total reward without observations.

As an example, suppose in our model, we observe the initial state with two *sprism* objects, object $\mathtt{o_1}$ at coordinate $(10, 35)$ and object $\mathtt{o_2}$ at coordinate $(12, 55)$. The goal is to have the two objects near each other, according to the previously defined `near` relation (centre-to-centre distance of less than $10cm$). We limit the sampled episode length to $d = 4$. Table 6.4 shows the output Q function estimated by running the planning algorithm for each of the applicable initial actions. We use the model shown in Appendix B. Therefore, the best first action to execute is: $\mathtt{push}(\mathtt{o_1}, \mathtt{left}, 25)$.

Table 6.4: Q function estimate obtained by running planning algorithm

Action	**Q function estimate**
$\mathtt{action}(\mathtt{push}(\mathtt{o_1}, \mathtt{left}, 15))$	-3.718
$\mathtt{action}(\mathtt{tap}(\mathtt{o_1}, \mathtt{left}, 10))$	-4.434
$\boldsymbol{action(push(o_1, left, 25))}$	$\mathbf{-2.402}$
$\mathtt{action}(\mathtt{tap}(\mathtt{o_1}, \mathtt{left}, 20))$	-4.918
$\mathtt{action}(\mathtt{tap}(\mathtt{o_1}, \mathtt{right}, 10))$	-4.224
$\mathtt{action}(\mathtt{tap}(\mathtt{o_1}, \mathtt{right}, 20))$	-4.999

6.6 Evaluation and Results

We want to investigate whether our learnt two-arms probabilistic relational affordance model can be used successfully in a table-top object manipulation setting, where a sequence of actions of the two arms is necessary to reach a goal.

For the experiments, we will use the iCub in a real setting. The iCub will use the perception capabilities to detect the shapes and positions of the objects on the table. After detecting the objects in the scene, the observations are sent to the planner, which infers the best action to execute towards the goal. The iCub executes this action. If the goal has not been reached, this process is continued. We limit in the planner the number of iCub actions to a sequence of maximum four actions. Figure 6.7 shows the data flow between the main software components, denoted as boxes.

Several software libraries and modules available from the iCub software repository were utilized and/or extended in order to learn the model and perform the experiments. The main software modules include the parametric actions module, the stereo vision, the image segmentation and the planner. Finally, there is a master module that controls the perception-planner-action loop, which runs while the goal has not been reached and the number of actions needed to reach the goal is less than four.

The parametric actions module is largely inspired on the actionsRenderingEngine[3] but defining an additional action, the two-hands push. In addition, all the actions were parameterised with the arm selection and hand displacement vector and follow the description of Section 6.5. Similar to the actionsRenderingEngine, our parametric actions module is built on top of the actionsPrimitive library[4] that allows to define actions as a set of end-effector points in the Cartesian space. The actionsPrimitive library provides interfaces to stop the execution if the forces on the arms exceed a threshold, which we utilize to stop the execution of the action in case of contacts with the table or other objects while reaching the pose prior to action execution. Actions move the arm over a preprogrammed distance defined by the parameter of the action.

The stereo vision module[5] utilizes the OpenCV[6] stereo algorithm presented in [Hirschmuller, 2008], considering the kinematic chain of the iCub and the mechanical mounting error of its eyes. After an initial calibration, the stereo module provides: stereo triangulation and pixel-to-3D mappings. The image

[3]https://github.com/robotology/icub-main/tree/master/src/modules/actionsRenderingEngine

[4]https://github.com/robotology/icub-main/tree/master/src/libraries/actionPrimitives

[5]https://github.com/robotology/stereo-vision

[6]http://opencv.org/

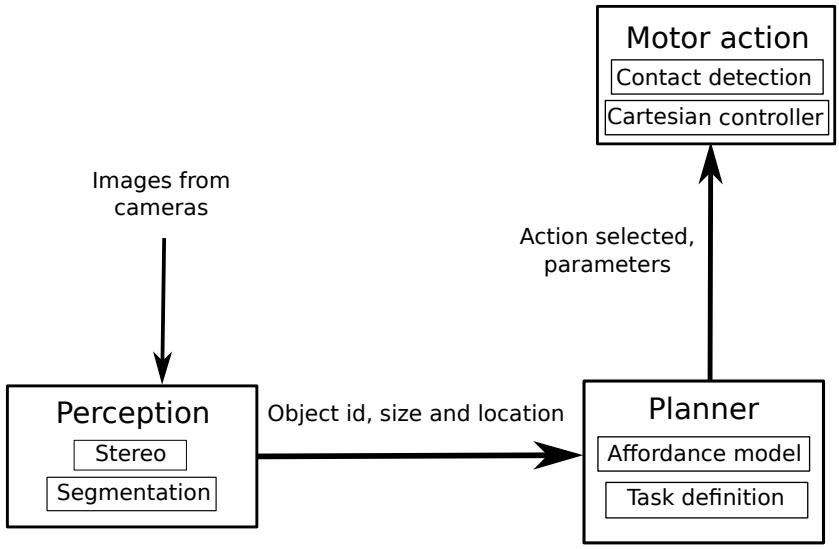


Figure 6.7: Main software components running for the experiments

segmentation module[7] provides a YARP [Metta et al., 2006] wrapper to the confidence-based edge detector and mean shift segmentation[8].

The planner module is a script written in the ProbLog extension for hybrid relational domains[9] that receives the input from the perception module in the form of clauses and outputs the action to be performed by the robot for a specific task, so each task has an associated script.

The master module is a set of Python[10] classes and a script that integrates the perception, planner and parametric actions in a loop that aims to reach the goal of the task while the goal is reachable in four steps.

Each experiment is set up as follows. The iCub is placed in front of a table with several objects on it in front of the robot, and the perception-planner-action loop is executed until the success or failure of the high-level goal. The robot

[7] https://svn.code.sf.net/p/robotcub/code/trunk/iCub/contrib/src/poeticon/poeticonpp/edisonSegmentationModule/

[8] http://coewww.rutgers.edu/riul/research/code/EDISON/

[9] https://dtai.cs.kuleuven.be/ml/systems/DC

[10] https://www.python.org/

is also given a high-level goal it needs to achieve. One such example of initial setting and given goal is:

Initial: $\texttt{shape}(\texttt{o}_1, \texttt{sprism}), \texttt{shape}(\texttt{o}_2, \texttt{bprism}), \texttt{shape}(\texttt{o}_3, \texttt{bprism}),$
$\texttt{coordX}(\texttt{o}_1, -22.3), \texttt{coordY}(\texttt{o}_1, 37.5),$
$\texttt{coordX}(\texttt{o}_2, -10.5), \texttt{coordY}(\texttt{o}_2, 43.2),$
$\texttt{coordX}(\texttt{o}_3, 9.6), \texttt{coordY}(\texttt{o}_3, 41.5)$
Goal: $\texttt{shape}(\texttt{Q}, \texttt{sprism}), \texttt{shape}(\texttt{R}, \texttt{bprism}), \texttt{near}(\texttt{Q}, \texttt{R})$

This initial configuration for the iCub can be seen in Figure 6.8(l). The final configuration shows the goal being reached successfully by placing the red object next to the orange object, where next means the distance between object centroids less than a predefined value (10cm).


Figure 6.8: Initial object placement (l), and final object locations (r)

For the tasks we assign to the robot, we will use an object placement game setting, inspired from [Lopes et al., 2007]. The high-level goal presented to the robot will be one of the following four, involving spatial relations between objects, in increasing order of complexity:

- Task 1. Place any two objects near each other
- Task 2. Place all *sprism* to the left of all *bprism*
- Task 3. Place two objects in the shelf, namely: a *sprism* and a *bprism*, where we define the shelf as being 40*cm* away from the robot on the y-axis. The *sprism* must be to the left of a *bprism*
- Task 4. Place all objects in the shelf, all *sprism* to the left of all *bprism*

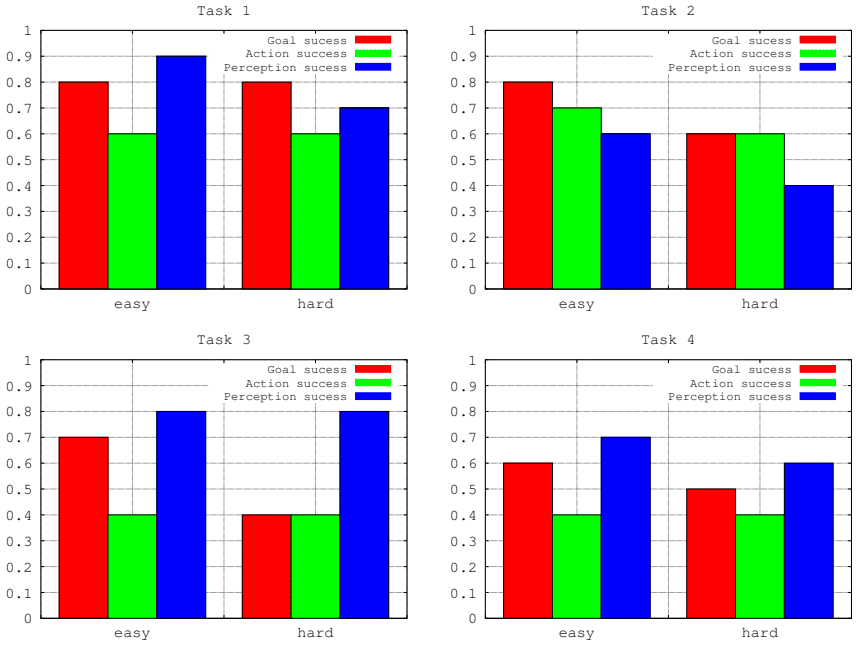


Figure 6.9: Performance by task. Task 1 is to place two of the objects in the world close to each other. Task 2 is to place all the small objects on the left of all middle size ones. Task 3 is to place two objects in the shelf, a small and a middle ones, being the small on the left side of the middle one. Task 4 is to place all the objects in shelf, placing all small on the left of all middle ones

Table 6.5: Consolidated results

	Goal success(%)	Average # of actions	Action success (%)	Perception success(%)
Easy	72.5	2.675	52.5	75
Hard	57.5	3.725	50	62.5
Easy + Hard	65	3.2	51.25	68.75

For each type of goal, we had 20 different object placements by the human, considering 10 easy configurations and 10 hard configurations, so 80 experiments were performed in total. The level of complexity is related to the number of objects on the table, namely, two objects for the easy configurations and three objects for the hard configurations. In addition to the success of the task to be done, we define the “Action Success” and the “Perception Success” for each

experiment.

The perception success is a binary value, being 1 if the object properties (object centroid, object size) were computed in a range of admissible values (+/- 2 cm) during the whole experiment. Any failure in perception during the experiment will lead to 0.

The "Action Success" is a binary value too, being 1 if the actions executed by the robot provided any of the effects learnt during the exploratory phase. The unexpected effects include objects tumbling down either by accident when reaching others or during the action execution. In addition, since the actions cause a translation of the objects with very small rotations during the exploratory phase, large object rotations (e.g., more than 30°) are considered as unexpected effects. If any unexpected effect occurs during the task execution, the action success becomes 0.

The "Action Success" along with the "Goal Success" evaluate the robustness of the decision making process to uncertainty in the action execution, while the "Perception Success" along with the "Goal Success" evaluate the robustness of the decision making to uncertainty in the perception. On one hand, Table 6.5 shows that the "Goal Success" is larger than the "Action Success" for the different subsets, so the perception and the planner modules are robust enough to deal with failures in the action execution. On the other hand, as expected the "Goal Success" is below the "Perception Success".

For a single individual action in the plan, the average motor success is 85.15%, while the perception of all objects correctly before the information is sent to the planner is successful in 82.03% of the cases. So both the motor action and perception are successful during one planning step only in 69.85% of the cases.

Figure 6.9 shows the performance measures marginalized by task and complexity, where it is important to remark the very low "Action Success" for the tasks 3 and 4. These two tasks require motion planning capabilities for action execution upon an object in order to avoid undesired contacts with other objects.

From these experiments, it can be seen that the iCub is able to reach the relational goal and achieve its planning task in most 65% of the settings, showing that the use of the two-arm relational affordance model in a planning setting can achieve results comparable to the ones presented in the previous chapters.

Experiments were run on computers with Intel Core $i5-2500$ $3.3GHz$ processors, $6MB$ cache, and $8GB$ memory. We implemented our model with DDCs and used 1000 samples for planning. Each planning step took about 2.5 seconds, while the action execution by itself for the iCub took 0.7 seconds.

Our approach has several limitations which we will introduce. All the limitations in our approach described in Chapters 4 and 5 concerning the relational affordances, experimental setup, and two-arm actions are also present here. The most notable one in this case is the limited number of objects. The reason for this is because given our objects, the maximum number of objects in the iCub's action space, leaving also room for the action manipulation, is three. We could have placed additional objects in the background that cannot be manipulated, as in the previous chapters, but the goal here was to showcase planning, and was concerned with considering all objects with applicable actions. A robot that is able to move around the table in order to increase its action space can be considered for future work.

6.7 Related Work

6.7.1 Affordances

Work related to this one includes research on tool use for robots, such as [Brown and Sammut, 2013] which learns the tool affordances of an object from a human demonstration together with a set of robot experimentations based on an inductive logic programming algorithm. Research on behaviour-grounded tool affordances such as [Stoytchev, 2005, Sinapov and Stoytchev, 2007] provides algorithms for the robot to learn the effects of its actions with given tools on other objects.

An overview of the several affordance formalisms and their relation to planning and robot control was presented in [Şahin et al., 2007]. Among the latest research, there's affordance learning for multi-step planning [Ugur et al., 2009], which tackles a related problem, but where the learnt affordance model is only for single objects single-arm actions, and where object relationships and interactions are not taken into account. Similar work on affordance frameworks includes [Cakmak et al., 2007] and [Steedman, 2002] among others, which are concerned with defining an affordance formalism to be used for control and planning, but without involving learning from demonstrations or a generalised modeling of multiple object interactions.

6.7.2 Robotics Manipulation

Research on two-arm robot manipulation includes research on learning, representing and generalising a task which presents a programming-by-demonstration framework for extracting and generalising knowledge about a

given task [Calinon et al., 2007], and similarly a programming-by-demonstration framework for dual-arm manipulation tasks [Zöllner et al., 2004]. There is also research on motion planning for dual-arm manipulation and re-grasping tasks [Vahrenkamp et al., 2009]. However, these do not use the concept of affordances, model or generalise over relations and interactions between manipulated objects and other objects in the environment, or build a two-arm manipulation model generalised from environment experimentation and the use of background knowledge.

Related to a table-top setting with multiple interacting objects, there is work on detecting [Kragic et al., 2005] and manipulating [Berenson and Srinivasa, 2008, Gienger et al., 2008, Jetchev and Toussaint, 2010] objects in cluttered environments, but this is usually concerned with detecting the objects and motion-planning for the arm in order to perform a grasp [Gienger et al., 2008, Jetchev and Toussaint, 2010], rather than creating a plan in order to solve a given task.

A related task as the one presented here, planning push actions for object placement on a cluttered table surface, is performed in simulation in [Cosgun et al., 2011] and with a PR-2 robot in [Emeli et al., 2011]. However, in these cases the object interactions are determined by a dynamics simulator based on the object physical properties, not taking uncertainty into account. This work does not generalise from two-object interactions to learn a model of a multiple-object setting.

6.7.3 Planning

This work is also related to the field of planning. In this area, STRIPS was introduced by R. Fikes and N. Nilsson [Fikes and Nilsson, 1971] for deterministic symbolic planning, as described in more detail in Section 2.4.2. A popular extension of STRIPS is the Planning Domain Definition Language (PDDL) developed by Drew McDermott [Mcdermott et al., 1998] and used in international planning competitions. These were later extended to a probabilistic domain.

Among the first probabilistic STRIPS planners were Buridan, presented in [Kushmerick et al., 1995], and Graphplan [Blum and Langford, 1998], followed by others. A probabilistic version of PDDL was also proposed [Younes and Littman, 2004]. There are several differences between DDC and probabilistic STRIPS, nonetheless, it is easy to map probabilistic STRIPS to DDC. Differently from probabilistic STRIPS, DDC does not have an explicit frame assumption (i.e. state not explicitly mentioned in the effects remains unchanged), in addition one DDC clause defines a class of random variables

at a time (e.g., $\texttt{pos(A)}_{t+1}$) and not the complete list of effects. Finally, all the random variables that are not defined by any DDC clause are implicitly removed from the state, thus an explicit definition of what to remove in the next state is not needed (delete list in STRIPS).

Building on probabilistic planning, there have been extensions to enable to model more complex noisy stochastic worlds, where a noise outcome implicitly models all rare and complex potential outcomes of rules. Some of this work building on noisy indeterministic deictic rules is also concerned among others with planning and learning STRIPS rules from data or from noisy observations [Pasula et al., 2004, Zettlemoyer et al., 2005]. Similarly based on noisy indeterministic deictic rules, in order plan object manipulation in a relational domain, is the work presented in [Lang and Toussaint, 2010a, Toussaint et al., 2010].

Other related work focuses on using event calculus for integrating planning and learning for an autonomous agent [Sablon and Bruynooghe, 1994] or learning action effects in partially observable domains [Mourão et al., 2010]. However, all these related works mentioned above are not done within an affordance framework which can learn and generalise from a robot motor babbling stage so they do not generalise from two-object interactions to a multiple object environment, or model the effects of actions performed with the other arm of a two-arm robot by symmetry. They do not map demonstrations to the action space of a robot and are rarely performed in a real-world robotics setting.

More related research in the context of planning is the work presented in [Lang and Toussaint, 2009]. Furthermore, action object complexes (OACs) [Wörgötter et al., 2009, Kruger et al., 2009] have been proposed for enabling efficient planning and execution of actions at all levels of the cognitive architecture, by combining STRIPS rules with the concept of affordances. Other research in this direction includes [Geib et al., 2006] and [Krüger et al., 2011], but this formalism doesn't involve the generalisation of multiple object interactions obtained from exploratory actions as done by relational affordances.

Other related work in decision-theoretic planning tackles similar problem domains [van Otterlo, 2009], but usually work with predefined and full action models, whereas we work with learned affordance models. Lastly, there is further research concerned with grounding planning operators by affordances [Lorken and Hertzberg, 2008] or an affordance formalism for planning from the perspective of user interfaces [Amant, 1999] among others.

6.7.4 Sample-based Planners

For our planning task we are not interested in solving an MDP entirely, but only finding the best first action from the current state. This simplifies the problem ignoring all the states that are not reachable from the current state in the remaining steps.

Our scenarios involves continuous random variables in the state, this generally requires a function approximation method to store the Q/V function (e.g. linear regression, see [Wiering and van Otterlo, 2012] for a review). However, the hybrid relational state space used here requires a non-trivial representation to store the Q function. This is beyond the scope of this research, thus we focused on simpler approaches. The class of planning algorithms closest to our needs are sampled-based planners that uses Monte-Carlo methods to estimate the optimal Q/V function. Some of the more notable examples include ε-soft on-policy Monte Carlo control [Sutton and Barto, 1998], sparse sampling [Kearns et al., 2002], UCT (Bandit based Monte-Carlo planning) [Kocsis and Szepesvári, 2006], and PRADA for noisy probabilistic relational rules [Lang and Toussaint, 2010b].

6.8 Conclusion and Future Work

We have presented in this chapter an extension of the relational affordance model for the continuous domain and two-arm actions in a sequential planning setting. The robot has to achieve a goal configuration in a table-top setting through a sequence of actions, and where acting on an object can produce effects on neighbouring interacting objects. Our model can be used for two-arm manipulation in a multiple object scene, as shown in our experiments.

The robot is able to reach fairly complex goals having a set of simple perceptual capabilities. In the future we will aim at defining more complex affordance models, which should include data from tactile perception during the babbling phase. During the task execution, we will investigate the addition of spatial relations from conventional motion planning and using other background rules for improving the object affordances outcome. The addition of those improvements to the relational affordances model and the planning algorithm will improve the goal success and allow the solution of more complex tasks.

Chapter 7

Occluded Object Search

This chapter[1] uses the concept of relational affordances to improve occluded object search performance. Before robots can tackle complex tasks, one of the main problems they face is finding the objects they need to manipulate. By learning and using a relational affordance model we can search for any of the multiple objects that afford a given action, each object type having a probability distribution over possible sizes and shapes, and where spatial relations between objects, such as co-occurrence and stacking, are modelled.

7.1 Introduction

The goal of robotics is to develop mobile, physical agents capable of reasoning, learning and manipulating their environment. To achieve a task, a robot first needs to find certain objects in the environment to manipulate. However, in a real life indoor environment, most objects are not immediately visible, but lie on shelves or cupboards behind other objects. If the searched object is not detected, the robot needs to reason about which objects to remove to continue its search.

Previous research in searching for occluded objects such as [Dogar et al., 2013, Wong et al., 2013] focused on object search in environments where all objects have known shapes and sizes, and lie on a flat surface. This is a practical approach in a known environment where only using a particular object can achieve a task. Imagine being in an unknown kitchen and wanting to drink

[1] This chapter presents work previously published in [Moldovan and De Raedt, 2014b]

water. We look at shelves and search for any object to use: it can be either a mug, or a glass, or maybe a plastic cup. Moreover, we do not know the exact shapes and sizes of these objects, although we posses prior knowledge of them. Finally, in this environment not all objects are on a flat surface: plates are stacked on top of each other, and the mug is on top of them.

A promising approach for the development of robots' skills is learning *object affordances*. We showed in the previous chapters that affordance models can be extended to relational affordances by modelling interactions and relations between objects with the help of statistical relational learning (SRL) [De Raedt and Kersting, 2008, De Raedt, 2008]. SRL was used there to generalize and enable inference in scenarios modelling spatial relations between multiple objects. In this chapter, we show how relational affordances can be used to search for an object that affords a given action.

7.1.1 Problem Statement and Approach

The robot is in a kitchen environment similar to [Wong et al., 2013], with the space partitioned in a finite set of disjoint containers with contents independent from each other, which we name shelves. Each shelf contains a set of objects, only some of which are visible to the robot, while the rest are occluded. The robot can remove the visible objects from a shelf, causing a subset of the previously occluded objects to become visible. Each object is associated with a set of actions it affords. The general relational affordance model is defined by a joint probability distribution over O, A, E, however here we are not interested in modelling the effects, which will be considered to be true if the action is executed, so in fact our relational affordance model will define a joint probability distribution over O and A. This joint probability distribution will be defined by a PPL program. Fig. 7.1 shows an example of a kitchen environment with multiple shelves. The robot's task is to find an object affording a given action a by removing as few objects as possible.

We make the following additional assumptions, similar to [Wong et al., 2013]: the environment is static, apart from any robot manipulation (i.e., object removal). Each shelf has known geometry. Each visible object is recognizable, so given a sufficiently clear view its type can be resolved without error. In addition to [Wong et al., 2013], we allow for objects to lie on other objects in a shelf (e.g., a stack of plates).

Let O_a represent the set of all objects affording action a. The robot proceeds as follows: it observes the set of shelves S. In each shelf $s_i \in S$ it notes the set of visible objects O_{vis_i}. For each s_i it computes, with the help of a relational affordance model, the probability that objects o in that shelf afford action

Figure 7.1: Kitchen shelves with visible and occluded objects.

a: $P(o \ in \ s_i, o \in O_a | O_{vis_i})$. The robot chooses the shelf with the highest probability and removes its visible objects, causing some occluded objects to become visible. The process is repeated until an object is found.

Compared to [Wong et al., 2013], relational affordance models allow a search setting where object shape and size are not fixed, but modelled by probability distributions, where more spatial relations can be easily modelled (e.g., stacking), and objects are associated to (and searched for) the actions they afford.

To build the relational affordance model, the robot is presented with:

(i) a set of objects, given by their properties (length, width, height, bounding shape and type label), and the list of afforded actions;

(ii) a set of images of shelves, where each object is labelled with its type, and from which a set of probabilities of object co-occurrences and stacking are extracted, and

(iii) background knowledge (i.e., logical rules) about object spatial relations and afforded actions.

Once this model is learnt, the occluded space is modelled, and then we can perform inference to compute the required probabilities.

7.1.2 Contributions and Outline

Previous work [Wong et al., 2013, Dogar et al., 2013] on searching for occluded objects focused on finding a specific given hidden object. The work of [Wong et al., 2013] uses object co-occurrence information and spatial constraints to build a model used to search for a specific object in a space populated by objects from a finite set of known types with fixed known shape and size. The main contribution of this chapter is using the concept of relational affordances to search for any object affording a given action. Multiple object types can afford the action, and each type allows for many different objects with size and shape modelled by probability distributions, thus relaxing some of the assumptions of previous work. Moreover, we allow for stacked objects, a more realistic modelling of objects in shelves, introducing more complex object spatial relations.

The use of SRL methods in relational affordances allows to easily model the object properties probability distributions and the relations to the actions they afford, as well as the spatial relations between objects (e.g., stacking) than individually modelling each constraint. Additionally, the model can be used as-is for different inference tasks without any need for additional modelling (e.g., computing the probability of two objects affording two different actions being on the same shelf, useful when moving between shelves is costly). Using simple affordance models using Bayesian Networks (BNs) as in [Lopes et al., 2007, Montesano et al., 2008] is infeasible in a setting with so many objects.

This chapter is organized as follows: Section 7.2 presents related work, Section 7.3 affordance-based models, and Section 7.4 the main contribution of this chapter: modelling and using relational affordances for occluded object search. Section 7.5 presents experimental results and we conclude in Section 7.6.

7.2 Related Work

Work on object search tackled manipulating sensors for better visibility of the searched object [Ye and Tsotsos, 1996, Shubina and Tsotsos, 2010, Sjö et al., 2009]. Object co-occurrence and object-scene context information for improving object search were explored in settings as [Kollar and Roy, 2009, Samadi et al., 2012, Schuster et al., 2012]. Using background knowledge from

previously seen, similar environments, to improve object search in unknown environments was studied in [Joho et al., 2011]. The use of spatial relations was studied for creating search strategies in multiple-room environments [Aydemir et al., 2011]. Exploiting domain knowledge for object discovery was considered in [Collet et al., 2013]. Learning hidden affordances based on object geometry and pose was studied in [Aldoma et al., 2012]. Lastly, systems such as KnowRob [Tenorth and Beetz, 2009] integrate knowledge representation and reasoning into general robotic control.

Recent work on occluded object search by manipulation includes a search algorithm for optimally finding a hidden object in a table-top setting [Dogar et al., 2013] and search for a hidden object in a setting with multiple containers with both visible and hidden objects [Wong et al., 2013], the latter being the setting we extend.

7.3 Affordance-based Models

Affordance models model the robot-world interaction by capturing action opportunities to structure the environment, e.g., a glass is handled in a different way than a book. An affordance is an intrinsic property of an object, allowing an action to be performed with the object by the agent performing the action [Rome et al., 2006]. For example, in Fig. 7.2 a newspaper, a book and a tablet afford reading, but out of these only a tablet affords playing a game.

Figure 7.2: Actions afforded by several objects

Affordances define relationships between the robot and the environment through the sensing and the motor capabilities of the robot [Lopes et al., 2007, Montesano et al., 2008]. When the robot explores its environment, it can build an affordance model to represent the correlations between the set of objects properties detected by its sensors, the repertoire of actions available to it, and the effects of performing the actions. However, perceiving an affordance does not mean the agent has to act upon it, acting being required only when an effect needs to be achieved [Rome et al., 2006].

In previous chapters, affordances were modelled as relations between the three variables: the set of object properties, the robot actions, and the set of effects. Such a generic affordance model was illustrated in Figure 3.1. In this chapter, we are less interested in modelling the effects of the actions, and more interested in modelling the relations between the set of object properties and the set of actions. The modelling of object properties such as size and shape will allow us to search for an object affording a given action.

Since we will not include the set of effects in our modelling of affordances in this chapter, we can consider these affordance models used for occluded object search as a special case of the more generic affordance models as shown in Figure 3.1. Indeed, in these probabilistic affordance models, one could have considered the set of effects of each action as being successful with a probability of 1.0 if the action is executed, and unsuccessful (probability of 0) if the action is not executed. For example, we can model the effect of the *water drinking* action as the water *being drunk* with a probability of 1.0. However, since these effects are not relevant to our inference task, they will not be explicitly modelled in the PPL program.

7.3.1 Single Object Affordances

Affordances are generally learned by the robot by exploring the environment and manipulating objects [Lopes et al., 2007, Montesano et al., 2008]. In our setting, we are interested in modelling the relations between object properties and the afforded actions. For showcasing our approach, we use the following object properties relevant to our search setting: length, width, height, shape (prism or cylinder) and type. Others can also be considered.

For learning the affordance model we use a set of IKEA kitchen objects taken from the on-line catalog[2]. We picked six object types from the corresponding IKEA categories: *glasses*, *cups*, *pitchers*, *plates*, *bowls* and *serving dishes*. We used 10 objects of each type, with the exception of pitchers from which only 5 were used. Fig. 7.3 shows all the 55 objects.

In our kitchen setting, we consider the six afforded actions: *water drinking*, *coffee drinking*, *pouring*, *dinner serving*, *soup serving* and *appetizer serving* (more could be added as well). Actions are afforded by objects from multiple types, e.g., water drinking is afforded by glasses as well as big cups. Table 7.1 outlines all actions and the objects that afford them.

In a real-world setting the robot is presented with a set of objects to observe for which it extracts the above mentioned properties. A system such as BLORT

[2]http://www.ikea.com/us/en/catalog/categories/departments/eating/

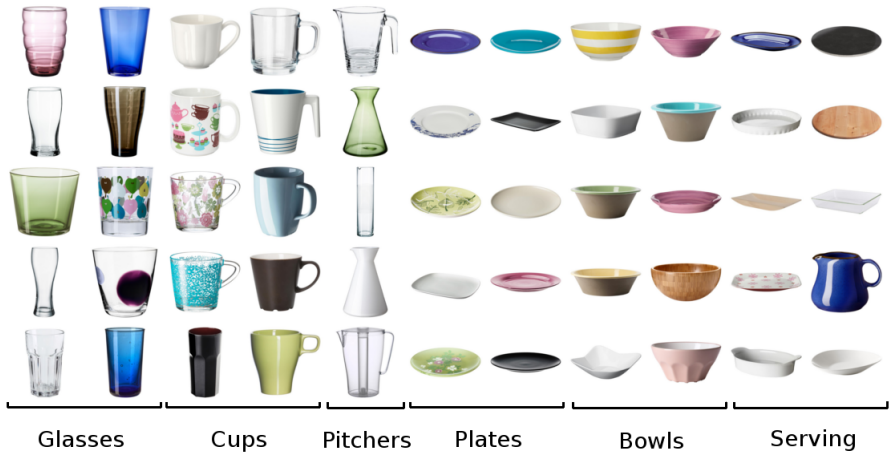


Figure 7.3: IKEA objects used for learning affordance model (Note: object images are not to same scale)

Table 7.1: Actions and the objects that afford them.

Action	**Objects**
Water drinking	Glasses Big cups (volume $> 25cl$)
Coffee drinking	Cups
Pouring	Pitchers Big glasses (volume $> 40cl$) Gravy boat serving dish (height $> 10cm$)
Dinner Serving	Plates Shallow bowls (height $< 5cm$) Small serving dishes (length $< 30cm$)
Soup Serving	Bowls
Appetizer Serving	Serving dishes (w/o gravy boat) Big plates (length $> 30cm$) Big, shallow bowls (length$> 30cm$,height$< 5cm$)

[Mörwald et al., 2010] can be used to model the objects by simple shapes. For each object, a human can tell the robot the afforded actions, taking into consideration the available motor capabilities of the robot. This list of afforded actions can be adjusted by exploratory manipulation of the objects by the robot. For ease of exposition, we assume the robot is presented directly with the set of

object properties and list of afforded actions for the 55 objects.

For representing the probability distributions of the object properties we learn and use a BN. In our setting, the conditional dependencies between the object properties are shown in Fig. 7.4. For the length and height we learn normal distributions for each object type from our 55 objects training data. For example, we obtained for the length of a glass $\mathcal{N}(6.05, 0.4742)$, for the height of a pitcher $\mathcal{N}(22.8, 4.6583)$, etc. (all numbers are cm). For the shape, we learn, depending on each type, the probability of being a prism or a cylinder, e.g., the shape of a plate has a probability of 83.33% of being a cylinder, and 16.67% a prism. The width is equal to the length if the shape is a cylinder, otherwise it is approximated with the normal distribution $\mathcal{N}(22.6364, 6.5463)$, whose parameters were similarly learnt from the training data. For modelling the actions, we use the rules from Table 7.1.

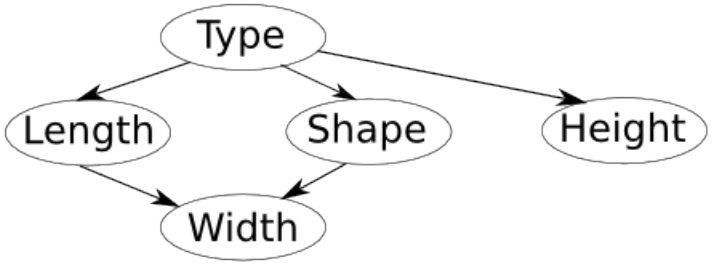


Figure 7.4: Conditional dependencies between object properties

We now have a single object affordance model for our setting. To extend it to a relational domain to be used for searching for occluded objects, we will model it with a PPL as in the previous chapters.

7.4 Relational Models for Affordances

In this section we describe our main contribution: modelling and using relational affordances for occluded object search. We have seen in the previous chapters how affordance models were extended with relations such as relative distance or angle of orientation between objects in order to perform action prediction in a multiple object environment. For searching for occluded objects, we are interested instead in extending the affordance model with the co-occurrence relation, as well as spatial relations (i.e., "on") between objects. Thus, as a difference to the approach from the previous chapters, we do not learn and then generalise a BN for a multiple object setting, but instead we model the

co-occurrence and stacking probabilities and relations, and use logical rules to restrict the spatial configurations of objects.

7.4.1 PPL Model

When using relational affordances for object search we need to deal with continuous distribution random variables (e.g, length, etc.), modelled by normal distributions. Distributional Clauses (DCs) [Gutmann et al., 2011b] are thus the better candidate for our statistical relational representation. In this chapter we will illustrate our approach using DCs. The overall model can be found in Appendix C.

For the modelling of affordances in PPL for the task in this chapter, the main predicates we will use are presented in Table 7.2 below.

We will start by modelling the probability distributions of the object properties. To model the length of a glass with probability distribution $\mathcal{N}(6.05, 0.4742)$, from the previous section, one can write:

$$\texttt{length}(\texttt{Obj}) \sim \texttt{gaussian}(6.05, 0.4742) \leftarrow \texttt{type}(\texttt{Obj}, \texttt{glass}).$$

with variable `Obj` universally quantified over the set of all objects; atom $\texttt{type}(\texttt{Obj}, \texttt{glass})$ being true if the type of `Obj` is a glass.

The probability distribution of the shape of a plate can be defined by:

$$\texttt{shape}(\texttt{Obj}) \sim \texttt{finite}([0.8333 : \texttt{cyl}, 0.1667 : \texttt{prism}]) \leftarrow \texttt{type}(\texttt{Obj}, \texttt{plate}).$$

We can encode the fact that the width of a cylinder object is the same as the length as follows:

$$\texttt{width}(\texttt{Obj}) \sim \texttt{finite}([1.0 : \texttt{L}]) \leftarrow\simeq (\texttt{shape}(\texttt{Obj})) == \texttt{cyl}, \texttt{L is } \simeq (\texttt{length}(\texttt{Obj})).$$

We define the actions the objects afford with the help of definite clauses. We illustrate this for the *pouring* action:

$$\begin{aligned}&\texttt{action}(\texttt{Obj}, \texttt{pour}) \leftarrow \texttt{type}(\texttt{Obj}, \texttt{pitcher}).\\ &\texttt{action}(\texttt{Obj}, \texttt{pour}) \leftarrow \texttt{type}(\texttt{Obj}, \texttt{glass}), \texttt{objVol}(\texttt{Obj}, \texttt{V}), \texttt{V} > 400.\\ &\texttt{action}(\texttt{Obj}, \texttt{pour}) \leftarrow \texttt{type}(\texttt{Obj}, \texttt{serving}), \simeq (\texttt{height}(\texttt{Obj})) > 10.\end{aligned}$$

where we previously defined atom $\texttt{objVol}(\texttt{Obj}, \texttt{V})$ to unify variable `V` to the volume in cm^3 of object `Obj`.

The rest of the affordance model is modelled using DCs in a similar manner.

Table 7.2: Predicates used for affordance modelling

Predicate	Meaning
type(Obj, ObjType)	The type of object Obj is ObjType. ObjType can be one of: glass, cup, pitcher, plate, bowl, serving.
shape(Obj)	Distribution of the shape of object Obj. Shape can be cyl or prism.
length(Obj)	Distribution of the length of object Obj.
width(Obj)	Distribution of the width of object Obj.
height(Obj)	Distribution of the height of object Obj.
action(Obj, AType)	The afforded action of object Obj is AType. AType can be one of the six defined action types.
observed_type(Obj)	Distribution of the type of an observed object Obj.
type_ifobs(Obj, ObservedType)	Distribution of the type of object Obj if we already observed another type ObservedType in the shelf.
ontype(Obj, ObservedType)	Distribution of the type of object Obj which sits on an object of type ObservedType.
objShorterThan(Obj, MaxH)	True if the height of object Obj is less than MaxH.
packedArea(Obj, AObj)	The packed area of object Obj is AObj.
objVolume(Obj, Value)	The volume of object Obj is Value.
area(Value, ObjectList)	The occluded area of size Value contains the objects in the list ObjectList.
stackOnTop(Obj, RemainH, StackObjList)	Stacks object Obj on top of a stack of objects given in StackObjList, where the available height remaining in the stack is RemainH.
checkStackConstraints (OBottom, OTop)	True if the length and width of the object on the bottom OBottom are greater or equal to the ones of the object on top OTop.

Once the program is defined, the inference algorithm based on sampling from [Gutmann et al., 2011b] or [Nitti et al., 2013] is used to compute the probability of a user's query.

7.4.2 Co-occurrence and Stacking Spatial Relations

Object type co-occurrence in a scene was used to improve object search results [Joho et al., 2011, Aydemir et al., 2011] and to create a generative model for searching for occluded objects [Wong et al., 2013]. We use it as one of the relations to extend our affordance model. If we consider an unknown object, which can equally afford any action, knowing it co-occurs on a shelf with another object of a known type makes the former object more likely to be of certain types, which in turn makes it more likely to afford certain actions. So co-occurrence of objects is an important relation to consider when modelling affordances.

There are several ways for a robot to learn co-occurrence probabilities of different object types. In an affordance setting, or a developmental framework, which propose acquiring skills by experimentation and interaction with the environment, the robot can extract these probabilities by observing many environments with shelves. Alternatively, such probabilities were obtained from the Web [Samadi et al., 2012], using semantic information obtained from search results. In our approach, we use labelled shelf images, which allows learning from realistic kitchen object distributions similar to a developmental framework approach. We used 66 shelves from 20 kitchen shelves images from Google Images. We labelled each object with its type, ignoring objects whose type was not one of the six modelled ones. Fig. 7.5 shows some of the shelves.

Figure 7.5: Sample shelf images from which co-occurrence and stacking probabilities were learnt

Since we want to find the shelf with the highest probability of having objects

with certain properties, we choose to model the concept of object co-occurrence by the probability of an object on a shelf being of a certain type given that on that shelf there are objects of another given type. We obtain probability values from the labelled shelf images by maximum likelihood estimation. For example, when observing a bowl:

$$\texttt{type\_ifobs}(\texttt{Obj}, \texttt{bowl}) \simeq \texttt{finite}([0.0798 : \texttt{glass}, 0.1007 : \texttt{cup}, \\ 0.0703 : \texttt{pitcher}, 0.3583 : \texttt{plate}, 0.3672 : \texttt{bowl}, \\ 0.0237 : \texttt{serving}]) \leftarrow \texttt{true}.$$

which means that in a shelf with bowl(s) there is a probability of 7.98% of an object `Obj` being a glass, and so on.

If the observed object types (later given as evidence) on the shelf are encoded by the random variables $\texttt{observed\_type}(\texttt{Obj})$, the type of the unknown occluded object `Obj` is given by the following logical rule:

$$\texttt{type}(\texttt{Obj}, \texttt{T}) \leftarrow\simeq (\texttt{observed\_type}(\texttt{Obj2})) == \texttt{T2}, \\ \simeq (\texttt{type\_ifobs}(\texttt{Obj}, \texttt{T2})) == \texttt{T}.$$

Object `Obj` has a probability $\texttt{type\_ifobs}(\texttt{Obj}, \texttt{T2})$ of being of type `T` whenever we observe an object of type `T2`.

Similarly, stacking probabilities are learnt and used. From the same shelf images, by maximum likelihood estimation, we learn the probability of an object being of a certain type given that it is on top of another object of a given type. For example, the probability for the object type on top of a plate:

$$\texttt{ontype}(\texttt{Obj}, \texttt{plate}) \simeq \texttt{finite}([0 : \texttt{glass}, 0.0024 : \texttt{cup}, \\ 0 : \texttt{pitcher}, 0.8676 : \texttt{plate}, 0.0284 : \texttt{bowl}, \\ 0 : \texttt{serving}, 0.1016 : \texttt{none}]) \leftarrow \texttt{true}.$$

where `none` signifies the probability that there is nothing on top of a plate. Then these stacking probabilities can be used in a logical rule similar to the one for co-occurrence.

7.4.3 Overall Model and Spatial Constraints

To use our defined model for inferring the probability that an object with the required affordance action is found in the occluded space we need to also model the occluded space.

We model the occluded space, with shape and size assumed known, with the help of logical rules. The packed area of an object is given by the `packedArea` predicate. The total area is defined recursively, assuming a greedy object

packing strategy. At the bottom we try to fit objects that are less than the given area, and we then decrease the remaining available area. Objects also have to satisfy the height requirement, and once an object is sampled we also keep track of the remaining height. Objects are stacked on top of each other as long as their total height is less than the maximum observed height. All objects are appended to a list. Different area modellings other than this greedy approach are also possible. Here is a simplified version of a model:

$$\begin{aligned}
&\texttt{area}(\texttt{AS}, []) \leftarrow \texttt{AS} <= 0.\\
&\texttt{area}(\texttt{AS}, \texttt{ObjList}) \leftarrow \texttt{objShorterThan}(\texttt{Obj}, \texttt{MaxHeight}),\\
&\texttt{RemainH is MaxHeight} - \simeq (\texttt{height}(\texttt{Obj})),\\
&\quad \texttt{stackOnTop}(\texttt{Obj}, \texttt{RemainH}, \texttt{StackObjList}),\\
&\quad \texttt{packedArea}(\texttt{Obj}, \texttt{AObj}),\ \texttt{NewA} = \texttt{AS} - \texttt{AObj},\\
&\quad \texttt{area}(\texttt{NewA}, \texttt{NewObjList}),\\
&\quad \texttt{append}([\texttt{Obj}|\texttt{StackObjList}], \texttt{NewObjList}, \texttt{ObjList}).
\end{aligned}$$

For modelling height constraints, we assumed vision sensor of the robot (e.g., camera) is at the same height as the shelf, so the maximum height of a stack of occluded objects is the maximum observed stack height in the shelf. Models where the camera looks at the shelf under an angle are also possible.

Spatial constraints can be modelled with logical rules. For example, bigger objects cannot be put on top of smaller ones:

$$\begin{aligned}
&\texttt{checkStackConstraints}(\texttt{OBottom}, \texttt{OTop}) \leftarrow\\
&\quad \simeq (\texttt{length}(\texttt{OTop})) <= \simeq (\texttt{length}(\texttt{OBottom})),\\
&\quad \simeq (\texttt{width}(\texttt{OTop})) <= \simeq (\texttt{width}(\texttt{OBottom})).
\end{aligned}$$

where the `checkStackConstraints` atom will be used in the body of the clause defining `stackOnTop`.

As stated before, the probability of objects o affording action a in a shelf s_i is given by $P(o\,in\,s_i, o \in O_a | O_{vis_i})$, where O_a is the set of all objects affording the action, and O_{vis_i} is the set of visible objects in s_i. To compute this probability using our relational affordance model, assuming an occluded space of size as in s_i, we use DCs with the inference methods of [Gutmann et al., 2011b], [Nitti et al., 2013] to compute the probability of the query:

$$\texttt{area}(\texttt{as}, \texttt{ObjList}), \texttt{member}(\texttt{Obj}, \texttt{ObjList}), \texttt{action}(\texttt{Obj}, \texttt{a}).$$

given the evidence formed by a conjunction of `observed_type` atoms.

At this point we have completed defining the model. The model we used for the evaluation of our approach can be found in Appendix C.

To compute which shelf is more likely to contain an occluded object which

affords the given action, we compute $\arg\max_{s_i} P(o \; in \; s_i, o \in O_a | O_{vis_i})$. This shelf is explored next by removing its visible objects.

Our relational model offers added flexibility since using the same model we can ask for the probability of an object on a shelf affording at least one of two different given actions, a_1, a_2, by asking the query:

$\texttt{area}(\texttt{as}, \texttt{ObjList}), \texttt{member}(\texttt{Obj}, \texttt{ObjList}), (\texttt{action}(\texttt{Obj}, \texttt{a}_1) \; ; \; \texttt{action}(\texttt{Obj}, \texttt{a}_2)).$

7.5 Evaluation and Results

We test our relational affordance model for occluded object search. We first show a small setting in simulation, then we investigate in larger settings how it compares with searching for a specific object (considering co-occurrence and spatial relations), and with baseline systematic searches.

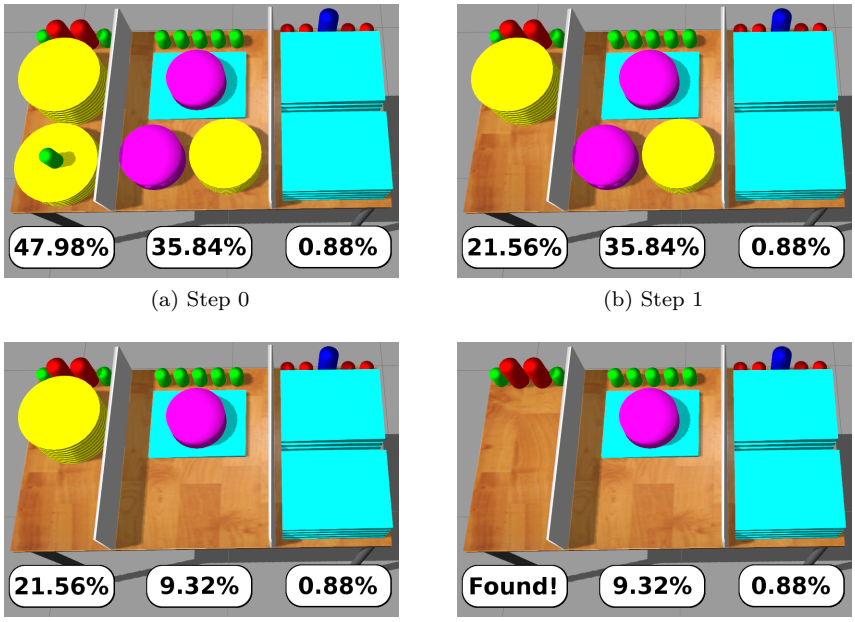
We integrated our model with the Gazebo robot simulator. We set up an environment with three shelves with objects, as in Fig. 7.6a. Objects are modelled by our two shapes. Object colours denote object type: glasses are red, cups green, pitchers blue, plates yellow, bowls magenta, serving dishes cyan. Only the front layer of objects (bottom of image) is visible to the robot, the rest is occluded. Similar to [Wong et al., 2013], our assumption is that a visible object type can be resolved without error, so we abstract the sensing by a program returning visible layer object types of a queried shelf.

Fig. 7.6 shows the step-by-step process of finding an object which affords pouring. This can be either one of the two big glasses in the back of the left shelf, or the pitcher in the back of the right shelf. The probabilities capture the likelihood that there is an object on that shelf affording pouring (since there can be more than one such object in all the shelves, or even none, these probabilities do not add up to one). Serving dishes very rarely co-occur with objects that afford pouring, and just a few (i.e., gravy boat) afford pouring themselves.

From initial setting (Fig. 7.6a), the search proceeds as follows:

1. Fig. 7.6a: the left shelf is the most likely to contain an object affording pouring. Its front layer is removed. No new object types are observed, while the available space decreases, causing the probability to drop.

2. Fig. 7.6b: the centre layer has the highest probability, its front layer is removed. Serving dishes are observed, which would barely increase probability, but the remaining available space, especially observed height, is greatly reduced, causing a big drop in the probability.



(a) Step 0

(b) Step 1

(c) Step 2

(d) Step 3

Figure 7.6: Simulation of search for an object affording pouring.

3. Fig. 7.6c: the left shelf is again the most likely one to contain such object, and its next layer is removed.
4. Fig. 7.6d: object affording pouring is found in left layer.

We investigate more thoroughly the benefits of using our model. The setting for this experiment is based on the one in [Wong et al., 2013]. Our environment was setup as follows: we picked 10 shelf images from Google Images (similar to Fig. 7.5, but with different object type distributions) not used for learning the model parameters, and labelled the object types in each. The shelves contain anywhere from 3 to 27 objects. We setup each shelf to have three different layers, the first visible, and the other two occluded behind it (similar to Fig. 7.6a).

From each of these shelf images, we obtain 10 different shelf configurations by randomly sampling each object into one of the three layers, while keeping the same stacking configuration. Then, for each object we sample a shape and size for it from our learned distributions, with the exception of stacked identical objects in the image, where we use the shape and size of the bottom object for the ones on top. So we generated a total of 100 shelves.

Our test setting consists of environments of 10 shelves, randomly picked each time from the 100 generated shelves. We pick an action to search for that is afforded by at least one occluded object, but by none of the initial visible ones (otherwise the search will terminate immediately). We generate and run 1000 different such object search scenarios.

We will compare our search approach using the relational affordance model, which searches for an object affording a given action considering the spatial relations we introduced, to three other search methods. The first one of these searches for a specific object taking into consideration the same spatial relations that we modelled in our relational affordance model. This search is equivalent to modelling the search in [Wong et al., 2013] together with the stacking and height constraints we defined in this chapter. The second search is a random search for an affordance, which picks shelfs at random at empties them until a target object affording the given action is found. We refer to this as a systematic search, to keep with the name convention of [Wong et al., 2013]. Finally, the third search is a random search for an exact object, which picks shelfs at random, and empties them until the exact object that we looked for is found.

In each scenario, the system uses one of its search strategies to pick the shelf to search next. Since we do not know where in an occluded layer the target might be, we remove the whole front layer to reveal the occluded objects behind it. The process continues until the target is found.

To summarize, the compared four search strategies are:

- **Relational affordance model**
- **Relational exact search**: use the model with spatial relations to look for the shelf most likely to have object with given type and shape, then in the layer search for specific object size (querying for size directly not possible since it is represented by continuous distributions)
- **Systematic affordance search**: shelves are chosen at random and emptied layer by layer until an object affording the given action is found
- **Systematic exact search**: shelves are chosen at random and emptied layer by layer until exact object found

The results are shown in Fig. 7.7. The figure shows a cumulative plot of the number of objects moved before the target is found. The more to the left a line is, the better.

For example, when using the relational affordance model, the median number of objects that need to be moved is 13, while 95% of the trials moved 32 objects or

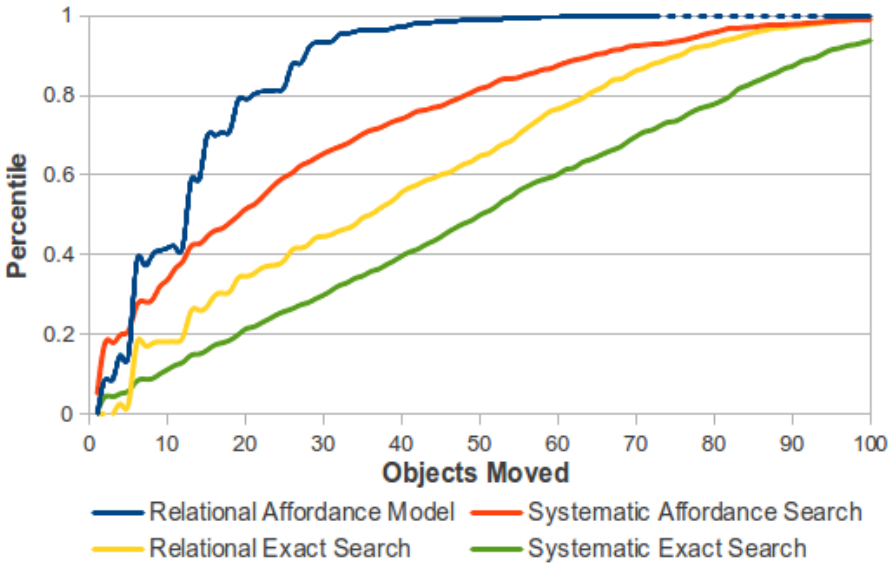


Figure 7.7: Comparison of the four search strategies: percentile of simulation trials finding an object within a number of moves

less. Using our relational affordance model is the best approach for minimising the number of objects moved. The second best approach is the systematic search for an affordance. Then comes the search for an exact object taking into account spatial object relations, and finally the systematic exact search. We can also deduce that incorporating object affordances into object search models is the most beneficial aspect, as the systematic search for an affordance performs better than a search for the exact object taking into account spatial relations.

Experiments were run on computers with Intel Core $i5-2500$ $3.3GHz$ processors, $6MB$ cache, and $8GB$ memory. We implemented our model with DCs and used 500 samples for computing query probabilities. Computing $P(o\,in\,s_i, o \in O_a|O_{vis_i})$ for one shelf s_i took on average 3.3 seconds.

Fig. 7.8(l) shows how run-time for obtaining one sample varies with the number of object types. We increased the number of types and actions by groups of six. If each action is afforded by only few objects, we need to increase the number of samples with the number of actions for the same relative precision, as each affordance now becomes more unlikely. Fig. 7.8(r) shows how run-time varies with the occluded area size for the 6 object types setting. Occluded shelf size for the experiments in Fig. 7.7 was around $0.2m^2$.

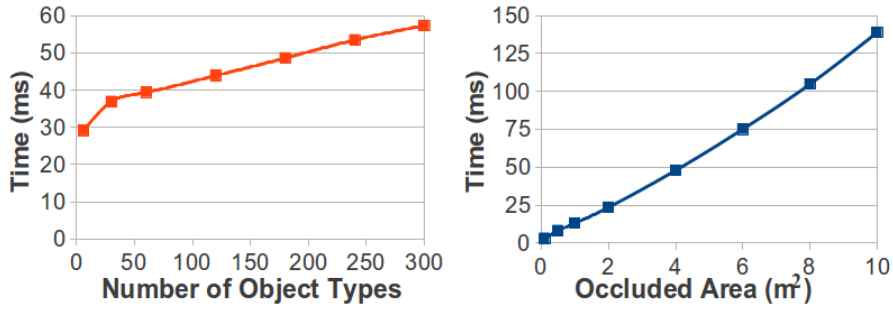


Figure 7.8: Run-time in milliseconds to obtain one sample while varying: (l) number of object types, (r) occluded search area

7.6 Conclusion and Future Work

In this chapter we presented an approach for learning and using a relational affordance model for occluded object search, and showed that it can be used successfully for improving search performance compared to searching for a specific object. As future work, we will model other spatial relations (e.g., "near": a salt shaker is likely near a pepper shaker), or organisational principles as in [Schuster et al., 2012]. Domain knowledge can be added for modelling other environments (e.g., living room). Uncertainty in the visible object types, or partial visibility constraints, can also easily be modelled.

Chapter 8

Conclusions and Future Work

We conclude the thesis with this chapter, which presents a summary of the work and a discussion about opportunities for future work.

8.1 Summary and Conclusions

To tackle real-world applications, robots need to deal with uncertainty, such as noisy sensors or modelling noisy actuators, but they also need to deal with higher level knowledge for reasoning and planning to achieve a task. In this thesis we investigated the use of statistical relational learning methods, which can be used to combine probabilistic and logical methods, to bridge this gap in robotics, especially in the area of affordance models.

Learning object affordances, following from the robotics developmental framework, has been used successfully lately for modelling and the development of humanoid robots' skills. Affordances model the robot-world interaction by capturing action opportunities to structure the robot's environment.

Learning relational affordance models was proposed as a way to extend the affordance concept using SRL methods to model multiple interacting objects in the environment. As opposed to previous methods of modelling affordances with BNs, the use of SRL models works for any number of objects in the scene, while also providing increased model comprehensibility. Multi-object scenes require expressive representation schemes to generalize over specific spatial configurations of objects and dealing with uncertainty and partial knowledge about the environment. We showed that a relational extension of the affordance

models, by learning from two object interactions, can be used for modeling a multi-object scene with success. We were able to learn a relational affordance model both in a discrete and in a continuous setting in a table-top environment.

The second goal of the thesis was to show that these learnt relational affordances can be successfully used in robotics **applications**.

An application we investigated was developing **models for two-arm robots**. We showed that using SRL methods we can create a relational affordance model for two-arm actions, for settings where these can be approximated by a combination of the two single-arm actions composing them. The arms may act simultaneously or sequentially, and the robot is given background knowledge about possible actions in its environment. We showed that such a model can be used successfully for two-arm manipulation in a multiple object scene, as shown by experiments on action recognition.

We investigated next the **planning of action sequences** for a table-top manipulation task. We used a relational affordance model in order to define a state transition model in a table-top setting, and provided the robot with a planning algorithm to be used to infer the next best possible action for the robot to execute towards reaching the given goal. We showed that the robot can then be successful in reaching its goal with the use of the relational affordance models.

Finally, in the field of **occluded object search**, we showed that relational affordances can be used successfully for improving search performance compared to searching for a specific object. The robot is able to search for any object affording a given action, where multiple object types can afford the action, with each type allowing for many different objects with size and shape modelled by probability distributions, and with more complex spatial relations such as the presence of stacked objects.

It can be noted throughout the thesis that learning and using relational affordance models for the various tasks involved several common steps. This approach can thus be reused for different tasks where we would need relational affordance models. The *methodology* of our approach can be generally summarised by the following steps:

1. Learn a **single object affordance** model. This model can then be used in multiple object settings when there are no object interactions. As we have seen in Chapter 4, this step is composed of the following:
    - Learning a BN model from single object babbling data
    - From this BN model build a PPL model

2. Learn a **multiple object affordance** model. As we have seen, this step is composed of the following:
    - Learning a BN model of two-object interactions babbling data, for discrete settings, or learning an LCG BN model for continuous settings
    - From the BN or LCG BN model, derive a PPL model by generalising by introducing variables for objects and adding background knowledge in the form of logical rules
3. For two-arm robots, build a **two-arm relational affordance** model in PPL, by generalising over the arms and adding rules constraining two-arm actions
4. For planning tasks, define a **state transition model** from the learnt relational affordance model.
5. Finally, once the models are learnt, an **inference task** is performed to solve the given task. This can be, but is not limited to, one of the following, for example:
    - Action prediction: $P(A|O, E)$
    - Object search
    - Planning

Therefore, a setting that requires the learning of relational affordances could be approached by following these steps. Additional steps can be added later, in cases where specific modelling is required for a particular task (e.g., adding rules about cutlery position relations in a table arrangement task).

We performed experiments that showed the validity of our models in a variety of settings. We used both simulation settings (iCub and PR2 robot) and settings with a real robot (iCub robot).

8.2 Future Work

We end the thesis with a short discussion about promising directions for future work. Relational affordance models can be used in a variety of areas. They are useful whenever there are interactions in the environment that the robot needs to model (e.g., acting on an object may affect other objects), or when the actions or effects of an object depend on its relations to other objects or the environment (e.g., a fork affords different actions if on the table or on

the ground). The modelling of relational affordances is useful in household environments.

Just as the regular affordance models, relational affordances can be used in any setting that requires one of the following for the robot: effect prediction, used to predict the outcome (and plan) an action, to recognize a performed action based on object properties and the effects of the action, or to select objects according to a task requirement. Therefore, the potential for the use of relational affordances is widespread.

There are several areas where there is potential for future work considering our affordance methodology. We can note that we first learn a BN or LCG BN from babbling data. This BN then determines the structure of the PPL program that defines our affordance model. Future work can consist of learning the structure of a PPL program directly from the babbling data. Also, in our methodology we add rules relevant for the specific task we are modelling. Future work can consist in analysing which rules contribute most to the accuracy of the models. Furthermore, the link between modelling with the help of rules and learning from babbling data can be investigated further. For example, one future direction might be to generate rules automatically from a small dataset. Or alternatively, while an initial modelling is done with the help of logical rules, data gathered afterwards can fine tune the model for the specific setting that the robot is investigating.

It can also be noted that most of our evaluation used a limited number of objects. This is because we used a fixed torso robot, which thus had a very limited action space for its arms, and thus it was only able to reach and act upon a very limited number of objects. Future work can consist of evaluating our approach with a robot that can freely move around the table, which would greatly increase its action space, and the number of objects it can reach. Also, in that case the robot might be able to act upon the objects from different directions, which can also be modelled.

Furthermore, each of the application areas investigated has a potential for future work.

In the area of two-arm modelling future work includes the investigation of the use of different additional spatial relations and using other background rules for two-arm actions, as well as more complex environments.

In the area of planning action sequences, future work includes modelling more complex environments, and going towards a full kitchen-robot scenario. There is also potential for future work in improving the planning algorithms for the robot to infer the sequence of actions needed to reach a goal.

In the area of occluded object search, future work includes modelling of other spatial relations (e.g., "near": a salt shaker is likely near a pepper shaker), or organisational principles as in [Schuster et al., 2012]. Domain knowledge can be added for modelling other environments (e.g., living room). Uncertainty in the visible object types, or partial visibility constraints, can also easily be modelled.

Appendix A

Discrete Action Prediction Relational Affordances Model

Below you can find an example of the relational affordance model for the six object action prediction task introduced in Chapter 4:

$\texttt{obj}(\texttt{o1})$.
$\texttt{obj}(\texttt{o2})$.
$\texttt{obj}(\texttt{o3})$.
$\texttt{obj}(\texttt{o4})$.
$\texttt{obj}(\texttt{o5})$.
$\texttt{obj}(\texttt{o6})$.

$0.5 :: \texttt{shape}(\texttt{Obj}, 1); 0.5 :: \texttt{shape}(\texttt{Obj}, 2) \leftarrow \texttt{obj}(\texttt{Obj})$.

$0.25 :: \texttt{initRelPos}(\texttt{o1}, \texttt{O2}, 1); 0.25 :: \texttt{initRelPos}(\texttt{o1}, \texttt{O2}, 2);$
$\quad 0.25 :: \texttt{initRelPos}(\texttt{o1}, \texttt{O2}, 3); 0.25 :: \texttt{initRelPos}(\texttt{o1}, \texttt{O2}, 4) \leftarrow \texttt{obj}(\texttt{O2}),$
$\quad \texttt{O2} \neq \texttt{o1}.$
$0.25 :: \texttt{initRelPos}(\texttt{o2}, \texttt{O2}, 1); 0.25 :: \texttt{initRelPos}(\texttt{o2}, \texttt{O2}, 2);$
$\quad 0.25 :: \texttt{initRelPos}(\texttt{o2}, \texttt{O2}, 3); 0.25 :: \texttt{initRelPos}(\texttt{o2}, \texttt{O2}, 4) \leftarrow \texttt{obj}(\texttt{O2}),$
$\quad \texttt{O2} \neq \texttt{o1}, \texttt{O2} \neq \texttt{o2}.$
$0.25 :: \texttt{initRelPos}(\texttt{o3}, \texttt{O2}, 1); 0.25 :: \texttt{initRelPos}(\texttt{o3}, \texttt{O2}, 2);$
$\quad 0.25 :: \texttt{initRelPos}(\texttt{o3}, \texttt{O2}, 3); 0.25 :: \texttt{initRelPos}(\texttt{o3}, \texttt{O2}, 4) \leftarrow \texttt{obj}(\texttt{O2}),$
$\quad \texttt{O2} \neq \texttt{o1}, \texttt{O2} \neq \texttt{o2}, \texttt{O2} \neq \texttt{o3}.$
$0.25 :: \texttt{initRelPos}(\texttt{o4}, \texttt{O2}, 1); 0.25 :: \texttt{initRelPos}(\texttt{o4}, \texttt{O2}, 2);$
$\quad 0.25 :: \texttt{initRelPos}(\texttt{o4}, \texttt{O2}, 3); 0.25 :: \texttt{initRelPos}(\texttt{o4}, \texttt{O2}, 4) \leftarrow \texttt{obj}(\texttt{O2}),$
$\quad \texttt{O2} \neq \texttt{o1}, \texttt{O2} \neq \texttt{o2}, \texttt{O2} \neq \texttt{o3}, \texttt{O2} \neq \texttt{o4}.$

```
0.25 :: initRelPos(o5, O2, 1); 0.25 :: initRelPos(o5, O2, 2);
        0.25 :: initRelPos(o5, O2, 3); 0.25 :: initRelPos(o5, O2, 4) ← obj(O2),
        O2 ≠ o1, O2 ≠ o2, O2 = o3, O2 ≠ o4, O2 ≠ o5.

% action
0.3333334 :: act(1); 0.3333333 :: act(2); 0.3333333 :: act(3) ← true.
0.1666667 :: actObj(o1); 0.1666667 :: actObj(o2); 0.1666666 :: actObj(o3);
        0.1666667 :: actObj(o4); 0.1666667 :: actObj(o5);
        0.1666666 :: actObj(o6) ← true.

% 1 object generalisations for displacement orientation
0.000001 :: dispOri(ObjMain, 1); 0.343749 :: dispOri(ObjMain, 2);
        0.531250 :: dispOri(ObjMain, 3); 0.062500 :: dispOri(ObjMain, 4);
        0.031250 :: dispOri(ObjMain, 5); 0.000000 :: dispOri(ObjMain, 6);
        0.031249 :: dispOri(ObjMain, 7); 0.000001 :: dispOri(ObjMain, 8)
        ← action(ObjMain, ObjSec, 1), initRelPosMag(ObjMain, ObjSec, 4).
0.000001 :: dispOri(ObjMain, 1); 0.000001 :: dispOri(ObjMain, 2);
        0.013887 :: dispOri(ObjMain, 3); 0.833333 :: dispOri(ObjMain, 4);
        0.138888 :: dispOri(ObjMain, 5); 0.000001 :: dispOri(ObjMain, 6);
        0.000001 :: dispOri(ObjMain, 7); 0.013888 :: dispOri(ObjMain, 8)
        ← action(ObjMain, ObjSec, 2), initRelPosMag(ObjMain, ObjSec, 4).
0.312498 :: dispOri(ObjMain, 1); 0.437499 :: dispOri(ObjMain, 2);
        0.000001 :: dispOri(ObjMain, 3); 0.124999 :: dispOri(ObjMain, 4);
        0.000001 :: dispOri(ObjMain, 5); 0.000001 :: dispOri(ObjMain, 6);
        0.000001 :: dispOri(ObjMain, 7); 0.124999 :: dispOri(ObjMain, 8)
        ← action(ObjMain, ObjSec, 3), initRelPosMag(ObjMain, ObjSec, 4).

% 1 object generalisations for displacement magnitude
1.0 :: dispMag(ObjSec, 1) ← action(ObjMain, ObjSec, 1),
        initRelPosMag(ObjMain, ObjSec, 4).
1.0 :: dispMag(ObjSec, 1) ← action(ObjMain, ObjSec, 2),
        initRelPosMag(ObjMain, ObjSec, 4).
1.0 :: dispMag(ObjSec, 1) ← action(ObjMain, ObjSec, 3),
        initRelPosMag(ObjMain, ObjSec, 4).

% from 2 object interaction BN :
0.000001 :: dispMag(ObjMain, 1); 0.152173 :: dispMag(ObjMain, 2);
        0.336957 :: dispMag(ObjMain, 3); 0.510869 :: dispMag(ObjMain, 4)
        ← shape(ObjMain, 1), action(ObjMain, _, 1).
0.000001 :: dispMag(ObjMain, 1); 0.509090 :: dispMag(ObjMain, 2);
        0.418182 :: dispMag(ObjMain, 3); 0.072727 :: dispMag(ObjMain, 4)
        ← shape(ObjMain, 1), action(ObjMain, _, 2).
0.000001 :: dispMag(ObjMain, 1); 0.007245 :: dispMag(ObjMain, 2);
        0.905797 :: dispMag(ObjMain, 3); 0.086957 :: dispMag(ObjMain, 4)
```

$\leftarrow$ shape(ObjMain, 1), action(ObjMain, _, 3).
0.011235 :: dispMag(ObjMain, 1); 0.337079 :: dispMag(ObjMain, 2);
0.325843 :: dispMag(ObjMain, 3); 0.325843 :: dispMag(ObjMain, 4)
$\leftarrow$ shape(ObjMain, 2), action(ObjMain, _, 1).
0.000001 :: dispMag(ObjMain, 1); 0.632352 :: dispMag(ObjMain, 2);
0.264706 :: dispMag(ObjMain, 3); 0.102941 :: dispMag(ObjMain, 4)
$\leftarrow$ shape(ObjMain, 2), action(ObjMain, _, 2).
0.007353 :: dispMag(ObjMain, 1); 0.014706 :: dispMag(ObjMain, 2);
0.117647 :: dispMag(ObjMain, 3); 0.860294 :: dispMag(ObjMain, 4)
$\leftarrow$ shape(ObjMain, 2), action(ObjMain, _, 3).

0.044199 :: dispOri(ObjMain, 1); 0.381215 :: dispOri(ObjMain, 2);
0.287293 :: dispOri(ObjMain, 3); 0.071823 :: dispOri(ObjMain, 4);
0.082873 :: dispOri(ObjMain, 5); 0.088398 :: dispOri(ObjMain, 6);
0.005525 :: dispOri(ObjMain, 7); 0.038674 :: dispOri(ObjMain, 8)
$\leftarrow$ action(ObjMain, _, 1).
0.000001 :: dispOri(ObjMain, 1); 0.000001 :: dispOri(ObjMain, 2);
0.008128 :: dispOri(ObjMain, 3); 0.642276 :: dispOri(ObjMain, 4);
0.349590 :: dispOri(ObjMain, 5); 0.000001 :: dispOri(ObjMain, 6);
0.000001 :: dispOri(ObjMain, 7); 0.000001 :: dispOri(ObjMain, 8)
$\leftarrow$ action(ObjMain, _, 2).
0.043795 :: dispOri(ObjMain, 1); 0.908759 :: dispOri(ObjMain, 2);
0.025547 :: dispOri(ObjMain, 3); 0.003650 :: dispOri(ObjMain, 4);
0.010949 :: dispOri(ObjMain, 5); 0.000001 :: dispOri(ObjMain, 6);
0.000001 :: dispOri(ObjMain, 7); 0.007297 :: dispOri(ObjMain, 8)
$\leftarrow$ action(ObjMain, _, 3).

0.337017 :: dispMag(ObjSec, 1); 0.187845 :: dispMag(ObjSec, 2);
0.116022 :: dispMag(ObjSec, 3); 0.359116 :: dispMag(ObjSec, 4)
$\leftarrow$ action(_, ObjSec, 1).
0.365853 :: dispMag(ObjSec, 1); 0.178862 :: dispMag(ObjSec, 2);
0.162602 :: dispMag(ObjSec, 3); 0.292683 :: dispMag(ObjSec, 4)
$\leftarrow$ action(_, ObjSec, 2).
0.726277 :: dispMag(ObjSec, 1); 0.131387 :: dispMag(ObjSec, 2);
0.032847 :: dispMag(ObjSec, 3); 0.109489 :: dispMag(ObjSec, 4)
$\leftarrow$ action(_, ObjSec, 3).

0.044199 :: dispOri(ObjSec, 1); 0.337017 :: dispOri(ObjSec, 2);
0.127072 :: dispOri(ObjSec, 3); 0.049724 :: dispOri(ObjSec, 4);
0.110497 :: dispOri(ObjSec, 5); 0.287293 :: dispOri(ObjSec, 6);
0.022099 :: dispOri(ObjSec, 7); 0.022099 :: dispOri(ObjSec, 8)
$\leftarrow$ action(_, ObjSec, 1).
0.121950 :: dispOri(ObjSec, 1); 0.097561 :: dispOri(ObjSec, 2);
0.065041 :: dispOri(ObjSec, 3); 0.260163 :: dispOri(ObjSec, 4);

$0.260163 :: \texttt{dispOri}(\texttt{ObjSec}, 5); 0.178862 :: \texttt{dispOri}(\texttt{ObjSec}, 6);$
$0.000001 :: \texttt{dispOri}(\texttt{ObjSec}, 7); 0.016259 :: \texttt{dispOri}(\texttt{ObjSec}, 8)$
$\leftarrow \texttt{action}(\_, \texttt{ObjSec}, 2).$
$0.124087 :: \texttt{dispOri}(\texttt{ObjSec}, 1); 0.197080 :: \texttt{dispOri}(\texttt{ObjSec}, 2);$
$0.072993 :: \texttt{dispOri}(\texttt{ObjSec}, 3); 0.087591 :: \texttt{dispOri}(\texttt{ObjSec}, 4);$
$0.116788 :: \texttt{dispOri}(\texttt{ObjSec}, 5); 0.164234 :: \texttt{dispOri}(\texttt{ObjSec}, 6);$
$0.083942 :: \texttt{dispOri}(\texttt{ObjSec}, 7); 0.153285 :: \texttt{dispOri}(\texttt{ObjSec}, 8)$
$\leftarrow \texttt{action}(\_, \texttt{ObjSec}, 3).$

$0.800000 :: \texttt{con}(\texttt{ObjMain}, \texttt{ObjSec}, 1); 0.200000 :: \texttt{con}(\texttt{ObjMain}, \texttt{ObjSec}, 2)$
$\leftarrow \texttt{initRelPosMag}(\texttt{ObjMain}, \texttt{ObjSec}, 1).$
$0.955000 :: \texttt{con}(\texttt{ObjMain}, \texttt{ObjSec}, 1); 0.045000 :: \texttt{con}(\texttt{ObjMain}, \texttt{ObjSec}, 2)$
$\leftarrow \texttt{initRelPosMag}(\texttt{ObjMain}, \texttt{ObjSec}, 2).$
$0.917431 :: \texttt{con}(\texttt{ObjMain}, \texttt{ObjSec}, 1); 0.082569 :: \texttt{con}(\texttt{ObjMain}, \texttt{ObjSec}, 2)$
$\leftarrow \texttt{initRelPosMag}(\texttt{ObjMain}, \texttt{ObjSec}, 3).$
$0.971264 :: \texttt{con}(\texttt{ObjMain}, \texttt{ObjSec}, 1); 0.028736 :: \texttt{con}(\texttt{ObjMain}, \texttt{ObjSec}, 2)$
$\leftarrow \texttt{initRelPosMag}(\texttt{ObjMain}, \texttt{ObjSec}, 4).$

$\texttt{initRelPosMag}(\texttt{O1}, \texttt{O2}, \texttt{D}) \leftarrow \texttt{initRelPos}(\texttt{O1}, \texttt{O2}, \texttt{D}), \texttt{O1} \neq \texttt{O2}.$
$\texttt{initRelPosMag}(\texttt{O1}, \texttt{O2}, \texttt{D}) \leftarrow \texttt{initRelPos}(\texttt{O2}, \texttt{O1}, \texttt{D}), \texttt{O1} \neq \texttt{O2}.$

$\texttt{action}(\texttt{O1}, \texttt{O2}, \texttt{A}) \leftarrow \texttt{actObj}(\texttt{O1}), \texttt{obj}(\texttt{O2}),$
$(\texttt{A} == 3 ->$
$\texttt{initRelPosMag}(\texttt{O1}, \texttt{O2}, \texttt{Dist}), \texttt{Dist} \neq 1;$
$\texttt{true}$
$),$
$\texttt{act}(\texttt{A}).$

Appendix B

Planning Continuous Relational Affordances Model

Below you can find an example of the relational affordance model for the planning task introduced in Chapter 6:

```
% initial observation
coord(ID) : 0 ~ val((X, Y)) ← observation(object(ID)) ~= (_, X, Y).

% state transition model
coord(ID) : t + 1 ~ val((X, Y)) ← \+ stop : t, coord(ID) : t ~= (Xt, Yt),
      displX(ID) : t ~= DXt, X is Xt + DXt,
      displY(ID) : t ~= DYt, Y is Yt + DYt.
coord(ID) : t + 1 ~ val((X, Y)) ← \+ stop : t, coord(ID) : t ~= (X, Y),
      \+ (displX(ID) : t ~= _), \+ (displY(ID) : t ~= _).
coord(ID) : t + 1 ~ val((X, Y)) ← \+ stop : t, coord(ID) : t ~= (X, Yt),
      \+ (displX(ID) : t ~= _), displY(ID) : t ~= DYt, Y is Yt + DYt.
coord(ID) : t + 1 ~ val((X, Y)) ← \+ stop : t, coord(ID) : t ~= (Xt, Y),
      displX(ID) : t ~= DXt, X is Xt + DXt, \+ (displY(ID) : t ~= _).
coord(ID) : t + 1 ~ val((X, Y)) ← \+ stop : t, \+ (policy : t + 1 ~= _),
      coord(ID) : t ~= (X, Y).

% auxiliary predicates
distX(ID1, ID2) : t ~ val(X) ← coord(ID1) : t ~= (X1, _),
      coord(ID2) : t ~= (X2, _), X is X2 − X1.
distY(ID1, ID2) : t ~ val(Y) ← coord(ID1) : t ~= (_, Y1),
      coord(ID2) : t ~= (_, Y2), Y is Y2 − Y1.
```

$$
\begin{aligned}
&\%\ \texttt{spatial relations}\\
&\texttt{near}(\texttt{ID1},\texttt{ID2}):\texttt{t} \leftarrow \texttt{coord}(\texttt{ID1}):\texttt{t} \sim= (\texttt{X1},\texttt{Y1}), \texttt{coord}(\texttt{ID2}):\texttt{t} \sim= (\texttt{X2},\texttt{Y2}),\\
&\qquad \texttt{ID1} \neq \texttt{ID2}, (\texttt{X2}-\texttt{X1})*(\texttt{X2}-\texttt{X1})+(\texttt{Y2}-\texttt{Y1})*(\texttt{Y2}-\texttt{Y1}) < 100.\\
&\texttt{left}(\texttt{ID1},\texttt{ID2}):\texttt{t} \leftarrow \texttt{coord}(\texttt{ID1}):\texttt{t} \sim= (\texttt{X1},\texttt{Y1}), \texttt{coord}(\texttt{ID2}):\texttt{t} \sim= (\texttt{X2},\texttt{Y2}),\\
&\qquad \texttt{ID1} \neq \texttt{ID2}, \texttt{X1} > \texttt{X2}.\\
&\texttt{right}(\texttt{ID1},\texttt{ID2}):\texttt{t} \leftarrow \texttt{coord}(\texttt{ID1}):\texttt{t} \sim= (\texttt{X1},\texttt{Y1}), \texttt{coord}(\texttt{ID2}):\texttt{t} \sim= (\texttt{X2},\texttt{Y2}),\\
&\qquad \texttt{ID1} \neq \texttt{ID2}, \texttt{X1} =< \texttt{X2}.\\
&\texttt{inshelf}(\texttt{ID1}):\texttt{t} \leftarrow \texttt{coord}(\texttt{ID1}):\texttt{t} \sim= (\texttt{X1},\texttt{Y1}), \texttt{Y1} > 40.\\
&\texttt{outshelf}(\texttt{ID1}):\texttt{t} \leftarrow \texttt{coord}(\texttt{ID1}):\texttt{t} \sim= (\texttt{X1},\texttt{Y1}), \texttt{Y1} =< 40.
\end{aligned}
$$

$$
\begin{aligned}
&\%\ \texttt{affordance model of main obj : displX, displY}\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(-0.465292, 0.005228) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{push}(\texttt{ID},\texttt{left},15)), \texttt{shape}(\texttt{ID},1).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(0.465292, 0.005228) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{push}(\texttt{ID},\texttt{right},15)), \texttt{shape}(\texttt{ID},1).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(-0.561257, 0.030335) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{push}(\texttt{ID},\texttt{left},15)), \texttt{shape}(\texttt{ID},2).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(0.561257, 0.030335) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{push}(\texttt{ID},\texttt{right},15)), \texttt{shape}(\texttt{ID},2).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(-1.537733, 0.182614) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{push}(\texttt{ID},\texttt{left},25)), \texttt{shape}(\texttt{ID},1).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(1.537733, 0.182614) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{push}(\texttt{ID},\texttt{right},25)), \texttt{shape}(\texttt{ID},1).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(-1.858147, 0.176808) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{push}(\texttt{ID},\texttt{left},25)), \texttt{shape}(\texttt{ID},2).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(1.858147, 0.176808) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{push}(\texttt{ID},\texttt{right},25)), \texttt{shape}(\texttt{ID},2).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(-6.501753, 0.346588) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID},\texttt{left},10)), \texttt{shape}(\texttt{ID},1).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(6.501753, 0.346588) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID},\texttt{right},10)), \texttt{shape}(\texttt{ID},1).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(-5.762988, 0.735104) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID},\texttt{left},10)), \texttt{shape}(\texttt{ID},2).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(5.762988, 0.735104) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID},\texttt{right},10)), \texttt{shape}(\texttt{ID},2).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(-15.736400, 0.510350) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID},\texttt{left},20)), \texttt{shape}(\texttt{ID},1).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(15.736400, 0.510350) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID},\texttt{right},20)), \texttt{shape}(\texttt{ID},1).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(-15.062937, 3.802915) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID},\texttt{left},20)), \texttt{shape}(\texttt{ID},2).\\
&\texttt{displX}(\texttt{ID}):\texttt{t} \sim \texttt{gaussian}(15.062937, 3.802915) \leftarrow\\
&\qquad \texttt{policy}:\texttt{t}+1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID},\texttt{right},20)), \texttt{shape}(\texttt{ID},2).
\end{aligned}
$$

```
displX(ID) : t ∼ gaussian(0.152475, 0.053337) ←
      policy : t + 1 ∼= action(push2hand(ID, both, 15)), shape(ID, 3).
displX(ID) : t ∼ gaussian(0.287867, 0.550603) ←
      policy : t + 1 ∼= action(push2hand(ID, both, 25)), shape(ID, 3).

displY(ID) : t ∼ gaussian(6.491981, 0.196423) ←
      policy : t + 1 ∼= action(push(ID, _, 15)), shape(ID, 1).
displY(ID) : t ∼ gaussian(6.024586, 0.232300) ←
      policy : t + 1 ∼= action(push(ID, _, 15)), shape(ID, 2).
displY(ID) : t ∼ gaussian(14.261647, 1.894410) ←
      policy : t + 1 ∼= action(push(ID, _, 25)), shape(ID, 1).
displY(ID) : t ∼ gaussian(13.909279, 2.101960) ←
      policy : t + 1 ∼= action(push(ID, _, 25)), shape(ID, 2).
displY(ID) : t ∼ gaussian(−0.546032, 0.243572) ←
      policy : t + 1 ∼= action(tap(ID, _, 10)), shape(ID, 1).
displY(ID) : t ∼ gaussian(−0.253518, 0.210861) ←
      policy : t + 1 ∼= action(tap(ID, _, 10)), shape(ID, 2).
displY(ID) : t ∼ gaussian(−0.049713, 0.223811) ←
      policy : t + 1 ∼= action(tap(ID, _, 20)), shape(ID, 1).
displY(ID) : t ∼ gaussian(−0.274637, 0.362116) ←
      policy : t + 1 ∼= action(tap(ID, _, 20)), shape(ID, 2).
displY(ID) : t ∼ gaussian(6.480125, 0.546778) ←
      policy : t + 1 ∼= action(push2hand(ID, _, 15)), shape(ID, 3).
displY(ID) : t ∼ gaussian(14.893133, 4.493807) ←
      policy : t + 1 ∼= action(push2hand(ID, _, 25)), shape(ID, 3).

% affordance model of sec obj : displX, displY
displX(ID2) : t ∼ gaussian(Mu, 0.112086) ←
      policy : t + 1 ∼= action(push(ID1, left, 15)), ID1 ≠ ID2,
      shape(ID1, 1), shape(ID2, 1),
      distX(ID1, ID2) : t ∼= DX, distY(ID1, ID2) : t ∼= DY,
      DX > −4.000000, DX < 4.000000, DY > 0.000000, DY < 23.000000,
      Mu is  − 0.017444 + 0.064572 ∗ DX + 0.008653 ∗ DY.
displX(ID2) : t ∼ gaussian(Mu, 0.112086) ←
      policy : t + 1 ∼= action(push(ID1, right, 15)), ID1 ≠ ID2,
      shape(ID1, 1), shape(ID2, 1),
      distX(ID1, ID2) : t ∼= DX, distY(ID1, ID2) : t ∼= DY,
      DX > −4.000000, DX < 4.000000, DY > 0.000000, DY < 23.000000,
      Mu is 0.017444 − 0.064572 ∗ DX − 0.008653 ∗ DY.
displX(ID2) : t ∼ gaussian(Mu, 0.139059) ←
      policy : t + 1 ∼= action(push(ID1, left, 15)), ID1 ≠ ID2,
      shape(ID1, 1), shape(ID2, 2),
      distX(ID1, ID2) : t ∼= DX, distY(ID1, ID2) : t ∼= DY,
```

$$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 26.500000,$$
$$\texttt{Mu is } 0.563939 + 0.093031 * \texttt{DX} - 0.010625 * \texttt{DY}.$$
$$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.139059) \leftarrow$$
$$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 15)), \texttt{ID1} \neq \texttt{ID2},$$
$$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),$$
$$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$$
$$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 26.500000,$$
$$\texttt{Mu is } -0.563939 - 0.093031 * \texttt{DX} + 0.010625 * \texttt{DY}.$$
$$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.038215) \leftarrow$$
$$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{left}, 15)), \texttt{ID1} \neq \texttt{ID2},$$
$$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),$$
$$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$$
$$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 26.500000,$$
$$\texttt{Mu is } -0.301250 + 0.040855 * \texttt{DX} + 0.013729 * \texttt{DY}.$$
$$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.038215) \leftarrow$$
$$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 15)), \texttt{ID1} \neq \texttt{ID2},$$
$$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),$$
$$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$$
$$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 26.500000,$$
$$\texttt{Mu is } 0.301250 - 0.040855 * \texttt{DX} - 0.013729 * \texttt{DY}.$$
$$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.190612) \leftarrow$$
$$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{left}, 15)), \texttt{ID1} \neq \texttt{ID2},$$
$$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 2),$$
$$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$$
$$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 30.000000,$$
$$\texttt{Mu is } 1.000917 + 0.096758 * \texttt{DX} - 0.031630 * \texttt{DY}.$$
$$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.190612) \leftarrow$$
$$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 15)), \texttt{ID1} \neq \texttt{ID2},$$
$$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 2),$$
$$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$$
$$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 30.000000,$$
$$\texttt{Mu is } -1.000917 - 0.096758 * \texttt{DX} + 0.031630 * \texttt{DY}.$$
$$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.229124) \leftarrow$$
$$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{left}, 25)), \texttt{ID1} \neq \texttt{ID2},$$
$$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 1),$$
$$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$$
$$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 33.000000,$$
$$\texttt{Mu is } -0.179056 - 0.030732 * \texttt{DX} + 0.008860 * \texttt{DY}.$$
$$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.229124) \leftarrow$$
$$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 25)), \texttt{ID1} \neq \texttt{ID2},$$
$$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 1),$$
$$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$$

$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 33.000000,$
$\texttt{Mu is } 0.179056 + 0.030732 * \texttt{DX} - 0.008860 * \texttt{DY}.$
$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.268269) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{left}, 25)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 36.500000,$
$\texttt{Mu is } 0.613436 - 0.028262 * \texttt{DX} - 0.014010 * \texttt{DY}.$
$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.268269) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 25)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 36.500000,$
$\texttt{Mu is } - 0.613436 + 0.028262 * \texttt{DX} + 0.014010 * \texttt{DY}.$
$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.176002) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{left}, 25)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 36.500000,$
$\texttt{Mu is } 0.614638 + 0.103497 * \texttt{DX} - 0.018070 * \texttt{DY}.$
$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.176002) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 25)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 36.500000,$
$\texttt{Mu is } - 0.614638 - 0.103497 * \texttt{DX} + 0.018070 * \texttt{DY}.$
$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.756077) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{left}, 25)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 2),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 40.000000,$
$\texttt{Mu is } 2.317712 + 0.124553 * \texttt{DX} - 0.061864 * \texttt{DY}.$
$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.756077) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 25)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 2),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 40.000000,$
$\texttt{Mu is } - 2.317712 - 0.124553 * \texttt{DX} + 0.061864 * \texttt{DY}.$
$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.409494) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 10)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 1),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$

$$
\begin{aligned}
&\texttt{DX} > -14.000000, \texttt{DX} < 0.000000, \texttt{DY} > -8.000000, \texttt{DY} < 8.000000,\\
&\texttt{Mu is } -7.054108 - 0.568435 * \texttt{DX} - 0.022926 * \texttt{DY}.\\
\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.409494) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 1),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > 0.000000, \texttt{DX} < 14.000000, \texttt{DY} > -8.000000, \texttt{DY} < 8.000000,\\
&\texttt{Mu is } 7.054108 + 0.568435 * \texttt{DX} + 0.022926 * \texttt{DY}.\\
\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.370471) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > -14.000000, \texttt{DX} < 0.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,\\
&\texttt{Mu is } -7.254029 - 0.588993 * \texttt{DX} + 0.029819 * \texttt{DY}.\\
\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.370471) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > 0.000000, \texttt{DX} < 14.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,\\
&\texttt{Mu is } 7.254029 + 0.588993 * \texttt{DX} - 0.029819 * \texttt{DY}.\\
\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.350041) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > -14.000000, \texttt{DX} < 0.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,\\
&\texttt{Mu is } -6.277036 - 0.521599 * \texttt{DX} - 0.034881 * \texttt{DY}.\\
\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.350041) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > 0.000000, \texttt{DX} < 14.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,\\
&\texttt{Mu is } 6.277036 + 0.521599 * \texttt{DX} + 0.034881 * \texttt{DY}.\\
\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.527925) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 2),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > -14.000000, \texttt{DX} < 0.000000, \texttt{DY} > -15.000000, \texttt{DY} < 15.000000,\\
&\texttt{Mu is } -7.650665 - 0.618647 * \texttt{DX} - 0.073816 * \texttt{DY}.\\
\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.527925) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 2),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},
\end{aligned}
$$

$\texttt{DX} > 0.000000, \texttt{DX} < 14.000000, \texttt{DY} > -15.000000, \texttt{DY} < 15.000000,$
$\texttt{Mu is } 7.650665 + 0.618647 * \texttt{DX} + 0.073816 * \texttt{DY}.$

$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.163209) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 20)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 1),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -24.000000, \texttt{DX} < 0.000000, \texttt{DY} > -8.000000, \texttt{DY} < 8.000000,$
$\texttt{Mu is } -19.407656 - 0.957144 * \texttt{DX} - 0.093876 * \texttt{DY}.$

$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.163209) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 20)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 1),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > 0.000000, \texttt{DX} < 24.000000, \texttt{DY} > -8.000000, \texttt{DY} < 8.000000,$
$\texttt{Mu is } 19.407656 + 0.957144 * \texttt{DX} + 0.093876 * \texttt{DY}.$

$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 1.623355) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 20)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -24.000000, \texttt{DX} < 0.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,$
$\texttt{Mu is } -15.006490 - 0.690665 * \texttt{DX} - 0.260981 * \texttt{DY}.$

$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 1.623355) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 20)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > 0.000000, \texttt{DX} < 24.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,$
$\texttt{Mu is } 15.006490 + 0.690665 * \texttt{DX} + 0.260981 * \texttt{DY}.$

$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.362709) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 20)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -24.000000, \texttt{DX} < 0.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,$
$\texttt{Mu is } -16.994002 - 0.872094 * \texttt{DX} - 0.210287 * \texttt{DY}.$

$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.362709) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 20)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > 0.000000, \texttt{DX} < 24.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,$
$\texttt{Mu is } 16.994002 + 0.872094 * \texttt{DX} + 0.210287 * \texttt{DY}.$

$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 6.888486) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 20)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 2),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$

$\texttt{DX} > -24.000000, \texttt{DX} < 0.000000, \texttt{DY} > -15.000000, \texttt{DY} < 15.000000,$
$\texttt{Mu is } -13.634903 - 0.535498 * \texttt{DX} - 0.098453 * \texttt{DY}.$
$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 6.888486) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 20)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 2),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > 0.000000, \texttt{DX} < 24.000000, \texttt{DY} > -15.000000, \texttt{DY} < 15.000000,$
$\texttt{Mu is } 13.634903 + 0.535498 * \texttt{DX} + 0.098453 * \texttt{DY}.$
$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.023930) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push2hand}(\texttt{ID1}, \texttt{both}, 15)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 3), \texttt{shape}(\texttt{ID2}, 1),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -15.000000, \texttt{DX} < 15.000000, \texttt{DY} > 0.000000, \texttt{DY} < 21.500000,$
$\texttt{Mu is } 0.611012 - 0.025087 * \texttt{DX} - 0.042556 * \texttt{DY}.$
$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.052449) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push2hand}(\texttt{ID1}, \texttt{both}, 15)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 3), \texttt{shape}(\texttt{ID2}, 2),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -15.000000, \texttt{DX} < 15.000000, \texttt{DY} > 0.000000, \texttt{DY} < 25.000000,$
$\texttt{Mu is } 0.634374 - 0.071267 * \texttt{DX} - 0.035268 * \texttt{DY}.$
$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.010000) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push2hand}(\texttt{ID1}, \texttt{both}, 25)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 3), \texttt{shape}(\texttt{ID2}, 1),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -15.000000, \texttt{DX} < 15.000000, \texttt{DY} > 0.000000, \texttt{DY} < 31.500000,$
$\texttt{Mu is } 0.686891 - 0.266406 * \texttt{DX} - 0.017621 * \texttt{DY}.$
$\texttt{displX}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.010000) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push2hand}(\texttt{ID1}, \texttt{both}, 25)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 3), \texttt{shape}(\texttt{ID2}, 2),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -15.000000, \texttt{DX} < 15.000000, \texttt{DY} > 0.000000, \texttt{DY} < 35.000000,$
$\texttt{Mu is } 0.686891 - 0.266406 * \texttt{DX} - 0.017621 * \texttt{DY}.$

$\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 1.193635) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{left}, 15)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 1),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 23.000000,$
$\texttt{Mu is } 9.206924 - 0.261416 * \texttt{DX} - 0.439812 * \texttt{DY}.$
$\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 1.193635) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 15)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 1),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$

$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 23.000000,$
$\texttt{Mu is } 9.206924 - 0.261416 * \texttt{DX} - 0.439812 * \texttt{DY}.$

$\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.614181) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{left}, 15)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 26.500000,$
$\texttt{Mu is } 11.321981 + 0.721621 * \texttt{DX} - 0.429613 * \texttt{DY}.$

$\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.614181) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 15)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 26.500000,$
$\texttt{Mu is } 11.321981 + 0.721621 * \texttt{DX} - 0.429613 * \texttt{DY}.$

$\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.683348) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{left}, 15)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 26.500000,$
$\texttt{Mu is } 7.700247 + 0.182275 * \texttt{DX} - 0.299870 * \texttt{DY}.$

$\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.683348) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 15)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 26.500000,$
$\texttt{Mu is } 7.700247 + 0.182275 * \texttt{DX} - 0.299870 * \texttt{DY}.$

$\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 1.351892) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{left}, 15)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 2),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 30.000000,$
$\texttt{Mu is } 15.430552 + 0.274448 * \texttt{DX} - 0.548803 * \texttt{DY}.$

$\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 1.351892) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 15)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 2),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$
$\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 30.000000,$
$\texttt{Mu is } 15.430552 + 0.274448 * \texttt{DX} - 0.548803 * \texttt{DY}.$

$\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 1.186698) \leftarrow$
$\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{left}, 25)), \texttt{ID1} \neq \texttt{ID2},$
$\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 1),$
$\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},$

$$
\begin{aligned}
&\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 33.000000,\\
&\texttt{Mu is } 17.642415 - 0.684587 * \texttt{DX} - 0.624928 * \texttt{DY}.\\
&\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 1.186698) \leftarrow\\
&\quad \texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 25)), \texttt{ID1} \neq \texttt{ID2},\\
&\quad \texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 1),\\
&\quad \texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\quad \texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 33.000000,\\
&\quad \texttt{Mu is } 17.642415 - 0.684587 * \texttt{DX} - 0.624928 * \texttt{DY}.\\
&\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.858723) \leftarrow\\
&\quad \texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{left}, 25)), \texttt{ID1} \neq \texttt{ID2},\\
&\quad \texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),\\
&\quad \texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\quad \texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 36.500000,\\
&\quad \texttt{Mu is } 25.757109 - 0.664403 * \texttt{DX} - 0.749699 * \texttt{DY}.\\
&\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.858723) \leftarrow\\
&\quad \texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 25)), \texttt{ID1} \neq \texttt{ID2},\\
&\quad \texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),\\
&\quad \texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\quad \texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 36.500000,\\
&\quad \texttt{Mu is } 25.757109 - 0.664403 * \texttt{DX} - 0.749699 * \texttt{DY}.\\
&\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 2.990827) \leftarrow\\
&\quad \texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{left}, 25)), \texttt{ID1} \neq \texttt{ID2},\\
&\quad \texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),\\
&\quad \texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\quad \texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 36.500000,\\
&\quad \texttt{Mu is } 23.337564 + 0.090889 * \texttt{DX} - 0.642734 * \texttt{DY}.\\
&\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 2.990827) \leftarrow\\
&\quad \texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 25)), \texttt{ID1} \neq \texttt{ID2},\\
&\quad \texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),\\
&\quad \texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\quad \texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 36.500000,\\
&\quad \texttt{Mu is } 23.337564 + 0.090889 * \texttt{DX} - 0.642734 * \texttt{DY}.\\
&\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 2.509630) \leftarrow\\
&\quad \texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{left}, 25)), \texttt{ID1} \neq \texttt{ID2},\\
&\quad \texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 2),\\
&\quad \texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\quad \texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 40.000000,\\
&\quad \texttt{Mu is } 26.761664 + 0.177639 * \texttt{DX} - 0.717811 * \texttt{DY}.\\
&\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 2.509630) \leftarrow\\
&\quad \texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{push}(\texttt{ID1}, \texttt{right}, 25)), \texttt{ID1} \neq \texttt{ID2},\\
&\quad \texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 2),\\
&\quad \texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{DX} > -4.000000, \texttt{DX} < 4.000000, \texttt{DY} > 0.000000, \texttt{DY} < 40.000000,\\
&\texttt{Mu is } 26.761664 + 0.177639 * \texttt{DX} - 0.717811 * \texttt{DY}.\\
\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.273516) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 1),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > -14.000000, \texttt{DX} < 0.000000, \texttt{DY} > -8.000000, \texttt{DY} < 8.000000,\\
&\texttt{Mu is } -1.242795 - 0.103245 * \texttt{DX} + 0.008034 * \texttt{DY}.\\
\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.273516) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 1),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > 0.000000, \texttt{DX} < 14.000000, \texttt{DY} > -8.000000, \texttt{DY} < 8.000000,\\
&\texttt{Mu is } -1.242795 - 0.103245 * \texttt{DX} + 0.008034 * \texttt{DY}.\\
\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.024880) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > -14.000000, \texttt{DX} < 0.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,\\
&\texttt{Mu is } -0.741759 - 0.069022 * \texttt{DX} + 0.032979 * \texttt{DY}.\\
\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.024880) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > 0.000000, \texttt{DX} < 14.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,\\
&\texttt{Mu is } -0.741759 - 0.069022 * \texttt{DX} + 0.032979 * \texttt{DY}.\\
\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.023871) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > -14.000000, \texttt{DX} < 0.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,\\
&\texttt{Mu is } -1.071702 - 0.091647 * \texttt{DX} + 0.001318 * \texttt{DY}.\\
\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.023871) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > 0.000000, \texttt{DX} < 14.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,\\
&\texttt{Mu is } -1.071702 - 0.091647 * \texttt{DX} + 0.001318 * \texttt{DY}.\\
\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.199776) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 2),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{DX} > -14.000000, \texttt{DX} < 0.000000, \texttt{DY} > -15.000000, \texttt{DY} < 15.000000,\\
&\texttt{Mu is } -1.279221 - 0.136356 * \texttt{DX} + 0.091869 * \texttt{DY}.\\
\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.199776) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 10)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 2),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > 0.000000, \texttt{DX} < 14.000000, \texttt{DY} > -15.000000, \texttt{DY} < 15.000000,\\
&\texttt{Mu is } -1.279221 - 0.136356 * \texttt{DX} + 0.091869 * \texttt{DY}.\\
\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.144167) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 20)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 1),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > -24.000000, \texttt{DX} < 0.000000, \texttt{DY} > -8.000000, \texttt{DY} < 8.000000,\\
&\texttt{Mu is } -0.495749 + 0.053124 * \texttt{DX} + 0.081786 * \texttt{DY}.\\
\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.144167) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 20)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 1),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > 0.000000, \texttt{DX} < 24.000000, \texttt{DY} > -8.000000, \texttt{DY} < 8.000000,\\
&\texttt{Mu is } -0.495749 + 0.053124 * \texttt{DX} + 0.081786 * \texttt{DY}.\\
\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.117223) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 20)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > -24.000000, \texttt{DX} < 0.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,\\
&\texttt{Mu is } -2.220463 - 0.080245 * \texttt{DX} - 0.046526 * \texttt{DY}.\\
\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 0.117223) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 20)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 1), \texttt{shape}(\texttt{ID2}, 2),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > 0.000000, \texttt{DX} < 24.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,\\
&\texttt{Mu is } -2.220463 - 0.080245 * \texttt{DX} - 0.046526 * \texttt{DY}.\\
\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 3.320578) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{left}, 20)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},\\
&\texttt{DX} > -24.000000, \texttt{DX} < 0.000000, \texttt{DY} > -11.500000, \texttt{DY} < 11.500000,\\
&\texttt{Mu is } -1.255358 - 0.026797 * \texttt{DX} + 0.099959 * \texttt{DY}.\\
\texttt{displY}(\texttt{ID2}) : \texttt{t} \sim \texttt{gaussian}(\texttt{Mu}, 3.320578) \leftarrow\\
&\texttt{policy} : \texttt{t} + 1 \sim= \texttt{action}(\texttt{tap}(\texttt{ID1}, \texttt{right}, 20)), \texttt{ID1} \neq \texttt{ID2},\\
&\texttt{shape}(\texttt{ID1}, 2), \texttt{shape}(\texttt{ID2}, 1),\\
&\texttt{distX}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DX}, \texttt{distY}(\texttt{ID1}, \texttt{ID2}) : \texttt{t} \sim= \texttt{DY},
\end{aligned}
$$

```
        DX > 0.000000, DX < 24.000000, DY > −11.500000, DY < 11.500000,
        Mu is  − 1.255358 − 0.026797 ∗ DX + 0.099959 ∗ DY.
displY(ID2) : t ∼ gaussian(Mu, 5.874461) ←
        policy : t + 1 ∼= action(tap(ID1, left, 20)), ID1 ≠ ID2,
        shape(ID1, 2), shape(ID2, 2),
        distX(ID1, ID2) : t ∼= DX, distY(ID1, ID2) : t ∼= DY,
        DX > −24.000000, DX < 0.000000, DY > −15.000000, DY < 15.000000,
        Mu is 2.423729 + 0.155799 ∗ DX + 0.211007 ∗ DY.
displY(ID2) : t ∼ gaussian(Mu, 5.874461) ←
        policy : t + 1 ∼= action(tap(ID1, right, 20)), ID1 ≠ ID2,
        shape(ID1, 2), shape(ID2, 2),
        distX(ID1, ID2) : t ∼= DX, distY(ID1, ID2) : t ∼= DY,
        DX > 0.000000, DX < 24.000000, DY > −15.000000, DY < 15.000000,
        Mu is 2.423729 + 0.155799 ∗ DX + 0.211007 ∗ DY.
displY(ID2) : t ∼ gaussian(Mu, 0.293392) ←
        policy : t + 1 ∼= action(push2hand(ID1, both, 15)), ID1 ≠ ID2,
        shape(ID1, 3), shape(ID2, 1),
        distX(ID1, ID2) : t ∼= DX, distY(ID1, ID2) : t ∼= DY,
        DX > −15.000000, DX < 15.000000, DY > 0.000000, DY < 21.500000,
        Mu is 7.043033 − 0.011891 ∗ DX − 0.510245 ∗ DY.
displY(ID2) : t ∼ gaussian(Mu, 0.267586) ←
        policy : t + 1 ∼= action(push2hand(ID1, both, 15)), ID1 ≠ ID2,
        shape(ID1, 3), shape(ID2, 2),
        distX(ID1, ID2) : t ∼= DX, distY(ID1, ID2) : t ∼= DY,
        DX > −15.000000, DX < 15.000000, DY > 0.000000, DY < 25.000000,
        Mu is 4.827165 + 0.001418 ∗ DX − 0.268838 ∗ DY.
displY(ID2) : t ∼ gaussian(Mu, 0.010000) ←
        policy : t + 1 ∼= action(push2hand(ID1, both, 25)), ID1 ≠ ID2,
        shape(ID1, 3), shape(ID2, 1),
        distX(ID1, ID2) : t ∼= DX, distY(ID1, ID2) : t ∼= DY,
        DX > −15.000000, DX < 15.000000, DY > 0.000000, DY < 31.500000,
        Mu is 11.673635 + 0.685020 ∗ DX − 0.514755 ∗ DY.
displY(ID2) : t ∼ gaussian(Mu, 0.010000) ←
        policy : t + 1 ∼= action(push2hand(ID1, both, 25)), ID1 ≠ ID2,
        shape(ID1, 3), shape(ID2, 2),
        distX(ID1, ID2) : t ∼= DX, distY(ID1, ID2) : t ∼= DY,
        DX > −15.000000, DX < 15.000000, DY > 0.000000, DY < 35.000000,
        Mu is 11.673635 + 0.685020 ∗ DX − 0.514755 ∗ DY.

% admissible actions
adm(action(push(ID, left, 15))) : t ← coord(ID) : t ∼= (X, Y),
        X > −10, X < 20, Y > 25, Y < 50.
adm(action(tap(ID, left, 10))) : t ← coord(ID) : t ∼= (X, Y),
```

```
      X > −15, X < 20, Y > 2, Y < 50.
adm(action(push(ID, left, 25))) : t ← coord(ID) : t ∼= (X, Y),
      X > −10, X < 20, Y > 25, Y < 50.
adm(action(tap(ID, left, 20))) : t ← coord(ID) : t ∼= (X, Y),
      X > −15, X < 20, Y > 2, Y < 50.

adm(action(push(ID, right, 15))) : t ← coord(ID) : t ∼= (X, Y),
      X > −20, X < 10, Y > 25, Y < 50.
adm(action(tap(ID, right, 10))) : t ← coord(ID) : t ∼= (X, Y),
      X > −20, X < 15, Y > 2, Y < 50.
adm(action(push(ID, right, 25))) : t ← coord(ID) : t ∼= (X, Y),
      X > −20, X < 10, Y > 25, Y < 50.
adm(action(tap(ID, right, 20))) : t ← coord(ID) : t ∼= (X, Y),
      X > −20, X < 15, Y > 2, Y < 50.

adm(action(push2hand(ID, both, 15))) : t ← shape(ID, 3),
      coord(ID) : t ∼= (X, Y), X > −10, X < 10, Y > 25, Y < 45.
adm(action(push2hand(ID, both, 25))) : t ← shape(ID, 3),
      coord(ID) : t ∼= (X, Y), X > −10, X < 10, Y > 25, Y < 45.

% policy
policy : t + 1 ∼ uniform(L) ←
      \ + action(_),
      findall_forward(action(A), adm(action(A)) : t, L),
      length(L, LL), LL > 0.
policy : t + 1 ∼ val(action(V)) ← action(V).

% stop criteria
stop : t ←
      shape(O1, 1), shape(O2, 2), left(O1, O2) : t, \ + outshelf(O) : t.
stop : t + 1 ← stop : t.

% reward
cumulreward : t + 1 ∼ val(New) ← cumulreward : t ∼= Old,
      \ + stop : t, New is Old − 1.
cumulreward : t + 1 ∼ val(Old) ← cumulreward : t ∼= Old, stop : t.
cumulreward : 0 ∼ val(0).

% plan predicate
go_plan(BestAction, OBS) ← retractall(shape(_, _)),
      %OBS = [observation(object(o1)) ∼= (1, −10, 35),
      %observation(object(o2)) ∼= (1, −12, 55)],
      (member(observation(object(ID)) ∼= (S, _, _), OBS),
      assert(shape(ID, S)), fail; true),
      bestaction(3, OBS, BestAction, 500), halt.
```

Appendix C

Object Search Relational Affordances Model

Below you can find an example of the occluded object search relational affordance model introduced in Chapter 7:

$\texttt{type}(\texttt{O}, \texttt{ground}) \leftarrow \texttt{GrID is Omod20}, \texttt{GrID} == 0.$

$\%$ model affordance BN
$\texttt{length}(\texttt{O}) \sim \texttt{finite}([0.0:1, 1.0:2, 0.0:3, 0.0:4, 0.0:5, 0.0:6,$
$0.0:7, 0.0:8, 0.0:9]) \leftarrow \texttt{type}(\texttt{O}, \texttt{glass}).$
$\texttt{length}(\texttt{O}) \sim \texttt{finite}([0.1176:1, 0.8824:2, 0.0:3, 0.0:4, 0.0:5, 0.0:6,$
$0.0:7, 0.0:8, 0.0:9]) \leftarrow \texttt{type}(\texttt{O}, \texttt{cup}).$
$\texttt{length}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.8:2, 0.2:3, 0.0:4, 0.0:5, 0.0:6,$
$0.0:7, 0.0:8, 0.0:9]) \leftarrow \texttt{type}(\texttt{O}, \texttt{pitcher}).$
$\texttt{length}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0:2, 0.0:3, 0.0:4, 0.0833:5, 0.5:6,$
$0.4167:7, 0.0:8, 0.0:9]) \leftarrow \texttt{type}(\texttt{O}, \texttt{plate}).$
$\texttt{length}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0:2, 0.0714:3, 0.4286:4, 0.0714:5, 0.1429:6,$
$0.2143:7, 0.0714:8, 0.0:9]) \leftarrow \texttt{type}(\texttt{O}, \texttt{bowl}).$
$\texttt{length}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0909:2, 0.0:3, 0.0:4, 0.0:5, 0.1818:6,$
$0.3636:7, 0.0909:8, 0.2727:9]) \leftarrow \texttt{type}(\texttt{O}, \texttt{serving}).$
$\texttt{length}(\texttt{O}) \sim \texttt{finite}([1:0]) \leftarrow \texttt{type}(\texttt{O}, \texttt{none}).$

$\texttt{height}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.2143:2, 0.5:3, 0.1429:4, 0.1429:5,$
$0.0:6]) \leftarrow \texttt{type}(\texttt{O}, \texttt{glass}).$
$\texttt{height}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.6471:2, 0.3529:3, 0.0:4, 0.0:5,$
$0.0:6]) \leftarrow \texttt{type}(\texttt{O}, \texttt{cup}).$

$\texttt{height}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0:2, 0.0:3, 0.6:4, 0.2:5,$
$0.2:6]) \leftarrow \texttt{type}(\texttt{O}, \texttt{pitcher}).$
$\texttt{height}(\texttt{O}) \sim \texttt{finite}([1.0:1, 0.0:2, 0.0:3, 0.0:4, 0.0:5,$
$0.0:6]) \leftarrow \texttt{type}(\texttt{O}, \texttt{plate}).$
$\texttt{height}(\texttt{O}) \sim \texttt{finite}([0.5:1, 0.2857:2, 0.2143:3, 0.0:4, 0.0:5,$
$0.0:6]) \leftarrow \texttt{type}(\texttt{O}, \texttt{bowl}).$
$\texttt{height}(\texttt{O}) \sim \texttt{finite}([0.8182:1, 0.0909:2, 0.0909:3, 0.0:4, 0.0:5,$
$0.0:6]) \leftarrow \texttt{type}(\texttt{O}, \texttt{serving}).$
$\texttt{height}(\texttt{O}) \sim \texttt{finite}([1:0]) \leftarrow \texttt{type}(\texttt{O}, \texttt{none}).$

$\texttt{shape}(\texttt{O}) \sim \texttt{finite}([1.0:\texttt{cyl}, 0.0:\texttt{prism}]) \leftarrow \texttt{type}(\texttt{O}, \texttt{glass}).$
$\texttt{shape}(\texttt{O}) \sim \texttt{finite}([1.0:\texttt{cyl}, 0.0:\texttt{prism}]) \leftarrow \texttt{type}(\texttt{O}, \texttt{cup}).$
$\texttt{shape}(\texttt{O}) \sim \texttt{finite}([1.0:\texttt{cyl}, 0.0:\texttt{prism}]) \leftarrow \texttt{type}(\texttt{O}, \texttt{pitcher}).$
$\texttt{shape}(\texttt{O}) \sim \texttt{finite}([0.8333:\texttt{cyl}, 0.1667:\texttt{prism}]) \leftarrow \texttt{type}(\texttt{O}, \texttt{plate}).$
$\texttt{shape}(\texttt{O}) \sim \texttt{finite}([0.7143:\texttt{cyl}, 0.2857:\texttt{prism}]) \leftarrow \texttt{type}(\texttt{O}, \texttt{bowl}).$
$\texttt{shape}(\texttt{O}) \sim \texttt{finite}([0.5455:\texttt{cyl}, 0.4545:\texttt{prism}]) \leftarrow \texttt{type}(\texttt{O}, \texttt{serving}).$
$\texttt{shape}(\texttt{O}) \sim \texttt{finite}([0.5:\texttt{cyl}, 0.5:\texttt{prism}]) \leftarrow \texttt{type}(\texttt{O}, \texttt{none}).$

$\texttt{width}(\texttt{O}) \sim \texttt{finite}([1.0:1, 0.0:2, 0.0:3, 0.0:4, 0.0:5, 0.0:6,$
$0.0:7, 0.0:8]) \leftarrow \texttt{length}(\texttt{O}) \sim= 1, \texttt{shape}(\texttt{O}) \sim= \texttt{cyl}.$
$\texttt{width}(\texttt{O}) \sim \texttt{finite}([0.0:1, 1.0:2, 0.0:3, 0.0:4, 0.0:5, 0.0:6,$
$0.0:7, 0.0:8]) \leftarrow \texttt{length}(\texttt{O}) \sim= 2, \texttt{shape}(\texttt{O}) \sim= \texttt{cyl}.$
$\texttt{width}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0:2, 1.0:3, 0.0:4, 0.0:5, 0.0:6,$
$0.0:7, 0.0:8]) \leftarrow \texttt{length}(\texttt{O}) \sim= 3, \texttt{shape}(\texttt{O}) \sim= \texttt{cyl}.$
$\texttt{width}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0:2, 0.0:3, 1.0:4, 0.0:5, 0.0:6,$
$0.0:7, 0.0:8]) \leftarrow \texttt{length}(\texttt{O}) \sim= 4, \texttt{shape}(\texttt{O}) \sim= \texttt{cyl}.$
$\texttt{width}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0:2, 0.0:3, 0.0:4, 1.0:5, 0.0:6,$
$0.0:7, 0.0:8]) \leftarrow \texttt{length}(\texttt{O}) \sim= 5, \texttt{shape}(\texttt{O}) \sim= \texttt{cyl}.$
$\texttt{width}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0:2, 0.0:3, 0.0:4, 0.0:5, 1.0:6,$
$0.0:7, 0.0:8]) \leftarrow \texttt{length}(\texttt{O}) \sim= 6, \texttt{shape}(\texttt{O}) \sim= \texttt{cyl}.$
$\texttt{width}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0:2, 0.0:3, 0.0:4, 0.0:5, 0.0:6,$
$1.0:7, 0.0:8]) \leftarrow \texttt{length}(\texttt{O}) \sim= 7, \texttt{shape}(\texttt{O}) \sim= \texttt{cyl}.$
$\texttt{width}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0:2, 0.0:3, 0.0:4, 0.0:5, 0.0:6,$
$0.0:7, 1.0:8]) \leftarrow \texttt{length}(\texttt{O}) \sim= 8, \texttt{shape}(\texttt{O}) \sim= \texttt{cyl}.$
$\texttt{width}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0:2, 0.0:3, 0.0:4, 0.0:5, 0.0:6,$
$0.0:7, 0.0:8]) \leftarrow \texttt{length}(\texttt{O}) \sim= 9, \texttt{shape}(\texttt{O}) \sim= \texttt{cyl}.$
$\texttt{width}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0:2, 0.0:3, 0.0:4, 0.0:5, 0.0:6,$
$0.0:7, 0.0:8]) \leftarrow \texttt{length}(\texttt{O}) \sim= 1, \texttt{shape}(\texttt{O}) \sim= \texttt{prism}.$
$\texttt{width}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0:2, 0.0:3, 0.0:4, 0.0:5, 0.0:6,$
$0.0:7, 0.0:8]) \leftarrow \texttt{length}(\texttt{O}) \sim= 2, \texttt{shape}(\texttt{O}) \sim= \texttt{prism}.$
$\texttt{width}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0:2, 0.0:3, 0.0:4, 0.0:5, 0.0:6,$
$0.0:7, 0.0:8]) \leftarrow \texttt{length}(\texttt{O}) \sim= 3, \texttt{shape}(\texttt{O}) \sim= \texttt{prism}.$
$\texttt{width}(\texttt{O}) \sim \texttt{finite}([0.0:1, 0.0:2, 0.0:3, 1.0:4, 0.0:5, 0.0:6,$

```
        0.0 : 7, 0.0 : 8]) ← length(O) ∼= 4, shape(O) ∼= prism.
width(O) ∼ finite([0.0 : 1, 0.0 : 2, 0.0 : 3, 0.0 : 4, 0.0 : 5, 0.0 : 6,
        0.0 : 7, 0.0 : 8]) ← length(O) ∼= 5, shape(O) ∼= prism.
width(O) ∼ finite([0.0 : 1, 0.0 : 2, 0.0 : 3, 1.0 : 4, 0.0 : 5, 0.0 : 6,
        0.0 : 7, 0.0 : 8]) ← length(O) ∼= 6, shape(O) ∼= prism.
width(O) ∼ finite([0.0 : 1, 0.0 : 2, 0.0 : 3, 0.0 : 4, 0.3333 : 5,
        0.3333 : 6, 0.3333 : 7, 0.0 : 8]) ← length(O) ∼= 7, shape(O) ∼= prism.
width(O) ∼ finite([0.0 : 1, 0.0 : 2, 0.0 : 3, 0.0 : 4, 0.0 : 5, 0.0 : 6,
        0.0 : 7, 0.0 : 8]) ← length(O) ∼= 8, shape(O) ∼= prism.
width(O) ∼ finite([0.0 : 1, 0.0 : 2, 0.0 : 3, 0.0 : 4, 0.3333 : 5,
        0.3333 : 6, 0.3333 : 7, 0.0 : 8]) ← length(O) ∼= 9, shape(O) ∼= prism.
width(O) ∼ finite([1 : 0]) ← type(O, none).

% helper area and volume predicates
objArea(O, A) ← length(O) ∼= L, width(O) ∼= W, A is L ∗ W.
objVol(O, V) ← shape(O) ∼= prism, length(O) ∼= L, width(O) ∼= W,
        height(O) ∼= H, V is L ∗ W ∗ H.
objVol(O, V) ← shape(O) ∼= cyl, length(O) ∼= L, height(O) ∼= H,
        V is 3.1416/4 ∗ L ∗ L ∗ H.

% define the affordance rules
action(O, water) ← type(O, glass).
action(O, water) ← type(O, cup), objVol(O, V), V > 9.
action(O, coffee) ← type(O, cup).
action(O, pour) ← type(O, pitcher).
action(O, pour) ← type(O, glass), objVol(O, V), V > 12.
action(O, pour) ← type(O, serving), height(O) ∼= H, H > 2.
action(O, dinner) ← type(O, plate).
action(O, dinner) ← type(O, bowl), height(O) ∼= 1.
action(O, dinner) ← type(O, serving), length(O) ∼= L, L < 6.
action(O, soup) ← type(O, bowl).
action(O, apetizer) ← type(O, serving).
action(O, apetizer) ← type(O, plate), length(O) ∼= L, L > 5.
action(O, apetizer) ← type(O, bowl), height(O) ∼= 1, length(O) ∼= L, L > 5.

% model co − occurence probabilities : cooccur(Obj, type2) ∼ type1
%probability of Obj being of type1 if on same shelf type2 is observed
cooccur(O, glass) ∼ finite([0.4331 : glass, 0.0859 : cup, 0.002 : pitcher,
        0.3285 : plate, 0.1505 : bowl, 0.0 : serving]) ← true.
cooccur(O, cup) ∼ finite([0.0780 : glass, 0.4341 : cup, 0.0751 : pitcher,
        0.2162 : plate, 0.1966 : bowl, 0.0 : serving]) ← true.
cooccur(O, pitcher) ∼ finite([0.0133 : glass, 0.1124 : cup, 0.0799 : pitcher,
        0.1680 : plate, 0.3896 : bowl, 0.0702 : serving]) ← true.
cooccur(O, plate) ∼ finite([0.0914 : glass, 0.0428 : cup, 0.0367 : pitcher,
```

```
        0.6050 : plate, 0.2232 : bowl, 0.0009 : serving]) ← true.
cooccur(O, bowl) ∼ finite([0.0798 : glass, 0.1007 : cup, 0.0703 : pitcher,
        0.3583 : plate, 0.3672 : bowl, 0.0237 : serving]) ← true.
cooccur(O, serving) ∼ finite([0.0 : glass, 0.0 : cup, 0.0263 : pitcher,
        0.2857 : plate, 0.2932 : bowl, 0.3947 : serving]) ← true.

observed_obj(X) ∼ uniform([glass, cup, pitcher, plate, bowl, serving])
        ← true.
type(O, T) ← GrID is Omod20, GrID = 1, observed_obj(OObs) ∼= TObs,
        cooccur(O, TObs) ∼= T.

% ontype(type, O) is the probability that the object
% on the given type is of a certain type
ontype(O, glass) ∼ finite([0.1429 : glass, 0 : cup, 0 : pitcher, 0 : plate,
        0 : bowl, 0 : serving, 0.8571 : none]) ← true.
ontype(O, cup) ∼ finite([0 : glass, 0.1566 : cup, 0 : pitcher, 0 : plate,
        0 : bowl, 0 : serving, 0.8434 : none]) ← true.
ontype(O, pitcher) ∼ finite([0 : glass, 0 : cup, 0 : pitcher, 0 : plate,
        0 : bowl, 0 : serving, 1 : none]) ← true.
ontype(O, plate) ∼ finite([0 : glass, 0.0024 : cup, 0 : pitcher,
        0.8676 : plate, 0.0284 : bowl, 0 : serving, 0.1016 : none]) ← true.
ontype(O, bowl) ∼ finite([0 : glass, 0 : cup, 0 : pitcher, 0 : plate,
        0.6476 : bowl, 0 : serving, 0.3524 : none]) ← true.
ontype(O, serving) ∼ finite([0 : glass, 0 : cup, 0 : pitcher, 0 : plate,
        0.0588 : bowl, 0.8235 : serving, 0.1177 : none]) ← true.

type(O, T) ← GrID is Omod20, GrID > 1, PrevO is O − 1,
        type(PrevO, PrevT), ontype(O, PrevT) ∼= T.

% define occluded area rules
area(A, MH, OID, TmpList, FinalList) ← A >= 0,
        height(OID) ∼= H, H =< MH, RemainH is MH − H,
        length(OID) ∼= Len, width(OID) ∼= W,
        objArea(OID, X), NewA is A − X,
        stackOn(NewA, MH, OID, Len, W, TmpList, FinalList, RemainH).

% init first ID to 1
area(A, MaxHeight, L) ← area(A, MaxHeight, 1, [], L).

% packing next object
packNext(A, MH, OID, L, FinalList) ← NewOID is (OID//20 + 1) ∗ 20 + 1,
        area(A, MH, NewOID, L, FinalList).

% stackOn(A, OID, NewOID, H) − area, old id, id to be used for
%the next stacking, remaining height
```

```
stackOn(A, MH, OID, Len, W, L, L, MaxH) ← A < 0.
stackOn(A, MH, OID, Len, W, L, FinalList, MaxH) ← A >= 0, MaxH < 0,
      packNext(A, MH, OID, L, FinalList).
stackOn(A, MH, OID, Len, W, L, FinalList, MaxH) ← A >= 0, MaxH >= 0,
      NewOID is OID + 1, type(NewOID, NewType),
      height(NewOID) ∼= H, RemainH is MaxH − H,
      length(NewOID) ∼= NewLen, width(NewOID) ∼= NewW,
      checkStackConstraints(A, MH, NewOID, NewType, Len, W, NewLen, NewW,
      [OID|L], FinalList, RemainH).

% stop if sampled length or width are bigger than object below
checkStackConstraints(A, MH, OID, NewType, Len, W, NewLen, NewW, L,
      FinalList, MaxH) ← NewLen > Len, packNext(A, MH, OID, L, FinalList).
checkStackConstraints(A, MH, OID, NewType, Len, W, NewLen, NewW, L,
      FinalList, MaxH) ← NewW > W, packNext(A, MH, OID, L, FinalList).
checkStackConstraints(A, MH, OID, none, Len, W, NewLen, NewW, L,
      FinalList, MaxH) ← packNext(A, MH, OID, L, FinalList).
checkStackConstraints(A, MH, NewOID, NewType, Len, W, NewLen, NewW, L,
      FinalList, MaxH) ← NewLen =< Len, NewW =< W, NewType ≠ none,
      stackOn(A, MH, NewOID, NewLen, NewW, L, FinalList, MaxH).


% affordance
affords(A_size, MH, Act) ← area(A_size, MH, L),
member(Obj, L), action(Obj, Act).

search_obj(A_size, MH, T, Sh, Len, W, H) ← area(A_size, MH, L),
      member(Obj, L), type(Obj, T), shape(Obj) ∼= Sh,
      length(Obj) ∼= Len, width(Obj) ∼= W, height(Obj) ∼= H.

go(N) ← write('result('), writePickedShelves, write(','),
      writeAff, write(','), writeObj, write(', ['),
      init, eval_query([observed_obj(1) ∼= serving,
      observed_obj(2) ∼= bowl, observed_obj(3) ∼= bowl], []
      query1_1, N, P1, _, _), write(P1), write(','), halt.
```

Bibliography

[Aksoy et al., 2011] Aksoy, E. E., Abramov, A., Dörr, J., Ning, K., Dellen, B., and Wörgötter, F. (2011). Learning the semantics of object-action relations by observation. *The International Journal of Robotics Research*.

[Aldoma et al., 2012] Aldoma, A., Tombari, F., and Vincze, M. (2012). Supervised learning of hidden and non-hidden 0-order affordances and detection in real scenes. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1732–1739. IEEE.

[Amant, 1999] Amant, R. S. (1999). Planning and user interface affordances. In *Proceedings of the 1999 International Conference on Intelligent User Interfaces*, pages 135–142.

[Aydemir et al., 2011] Aydemir, A., Sjöö, K., Folkesson, J., Pronobis, A., and Jensfelt, P. (2011). Search in the real world: Active visual object search based on spatial relations. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2818–2824. IEEE.

[Bancilhon and Ramakrishnan, 1986] Bancilhon, F. and Ramakrishnan, R. (1986). *An amateur's introduction to recursive query processing strategies*, volume 15. ACM.

[Berenson and Srinivasa, 2008] Berenson, D. and Srinivasa, S. S. (2008). Grasp synthesis in cluttered environments for dexterous hands. In *Humanoid Robots, 2008. 8th IEEE-RAS International Conference on*, pages 189–196. IEEE.

[Bernstein, 1967] Bernstein, N. A. (1967). The co-ordination and regulation of movements.

[Bertoli et al., 2001] Bertoli, P., Cimatti, A., Roveri, M., and Traverso, P. (2001). Planning in nondeterministic domains under partial observability via symbolic model checking. In *IJCAI*, volume 2001, pages 473–478.

[Bibel et al., 1989] Bibel, W., del Cerro, L. F., Fronhöfer, B., and Herzig, A. (1989). Plan generation by linear proofs: on semantics. In *GWAI-89 13th German Workshop on Artificial Intelligence*, pages 49–62. Springer.

[Blum and Langford, 1998] Blum, A. L. and Langford, J. C. (1998). Probabilistic planning in the graphplan framework.

[Brown and Sammut, 2013] Brown, S. and Sammut, C. (2013). A relational approach to tool-use learning in robots. In *Inductive Logic Programming*, pages 1–15.

[Browne et al., 2012] Browne, C., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43.

[Cakmak et al., 2007] Cakmak, M., Dogar, M. R., Ugur, E., and Sahin, E. (2007). Affordances as a framework for robot control. In *International Conference on Epigenetic Robotics (EpiRob)*.

[Calinon et al., 2007] Calinon, S., Guenter, F., and Billard, A. (2007). On learning, representing, and generalizing a task in a humanoid robot. *Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(2):286–298.

[Chemero, 2003] Chemero, A. (2003). An outline of a theory of affordances. *Ecological psychology*, 15(2):181–195.

[Chemero and Turvey, 2007] Chemero, A. and Turvey, M. T. (2007). Gibsonian affordances for roboticists. *Adaptive Behavior*, 15(4):473–480.

[Chickering, 1996] Chickering, D. M. (1996). Learning bayesian networks is np-complete. In *Learning from data*, pages 121–130. Springer.

[Christoudias et al., 2002] Christoudias, C. M., Georgescu, B., and Meer, P. (2002). Synergism in low level vision. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 4, pages 150–155. IEEE.

[Cocora et al., 2006] Cocora, A., Kersting, K., Plagemann, C., Burgard, W., and De Raedt, L. (2006). Learning relational navigation policies. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2792–2797. IEEE.

[Collet et al., 2013] Collet, A., Xiong, B., Gurau, C., Hebert, M., and Srinivasa, S. (2013). Exploiting domain knowledge for object discovery. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*.

[Coradeschi and Saffiotti, 2003] Coradeschi, S. and Saffiotti, A. (2003). An introduction to the anchoring problem. *Robotics and Autonomous Systems*, 43(2-3):85–96.

[Cos-Aguilera et al., 2003] Cos-Aguilera, I., Canamero, L., and Hayes, G. (2003). Learning object functionalities in the context of behaviour selection. In *Proceedings of the 4th British Conference Towards Intelligent Autonomous Robots*.

[Cos-Aguilera et al., 2005] Cos-Aguilera, I., Canamero, L., and Hayes, G. (2005). Motivation driven learning of action affordances. In *Procs of Symposium on Agents that Want and Like-Motivational and Emotional Roots of Cognition and Action*. SSAISB.

[Cosgun et al., 2011] Cosgun, A., Hermans, T., Emeli, V., and Stilman, M. (2011). Push planning for object placement on cluttered table surfaces. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*.

[Croonenborghs et al., 2007] Croonenborghs, T., Ramon, J., Blockeel, H., and Bruynooghe, M. (2007). Online learning and exploiting relational models in reinforcement learning. In *IJCAI*, volume 2007, pages 726–731.

[Şahin et al., 2007] Şahin, E., Çakmak, M., Doğar, M. R., Uğur, E., and Üçoluk, G. (2007). To afford or not to afford: A new formalization of affordances towards affordance based robot control. *Adaptive Behavior*, 15(4):447–472.

[De Raedt, 2008] De Raedt, L. (2008). *Logical and Relational Learning*. Springer.

[De Raedt and Kersting, 2008] De Raedt, L. and Kersting, K. (2008). Probabilistic inductive logic programming. In *Probabilistic Inductive Logic Programming*, pages 1–27.

[De Raedt et al., 2007] De Raedt, L., Kimmig, A., and Toivonen, H. (2007). Problog: A probabilistic Prolog and its application in link discovery. In *IJCAI*, volume 7, pages 2462–2467.

[Demiris and Dearden, 2005] Demiris, Y. and Dearden, A. (2005). From motor babbling to hierarchical learning by imitation: a robot developmental pathway.

[Dempster et al., 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38.

[Detry et al., 2009] Detry, R., Baseski, E., Popovic, M., Touati, Y., Kruger, N., Kroemer, O., Peters, J., and Piater, J. (2009). Learning object-specific grasp affordance densities. In *Development and Learning, 2009. ICDL 2009. IEEE 8th International Conference on*, pages 1–7. IEEE.

[Detry et al., 2010] Detry, R., Kraft, D., Buch, A. G., Kruger, N., and Piater, J. (2010). Refining grasp affordance models by experience. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2287–2293. IEEE.

[Dogar et al., 2013] Dogar, M., Koval, M., Tallavajhula, A., and Srinivasa, S. (2013). Object search by manipulation. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*.

[Emeli et al., 2011] Emeli, V., Kemp, C., and Stilman, M. (2011). Push planning for object placement in clutter using the PR-2. In *IROS PR2 Workshop*.

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. (1971). STRIPS: A new approach to the application theorem proving to problem solving. *Artificial Intelligence*, 5(2):189–208.

[Flach, 1994] Flach, P. A. (1994). *Simply logical: intelligent reasoning by example*. John Wiley & Sons, Inc.

[Flores et al., 2011] Flores, M. J., Nicholson, A. E., Brunskill, A., Korb, K. B., and Mascaro, S. (2011). Incorporating expert knowledge when learning bayesian network structure: a medical case study. *Artificial intelligence in medicine*, 53(3):181–204.

[Fritz et al., 2006] Fritz, G., Paletta, L., Breithaupt, R., Rome, E., and Dorffner, G. (2006). Learning predictive features in affordance based robotic perception systems. In *Intelligent Robots and Systems (IROS), 2006 IEEE/RSJ International Conference on*, pages 3642–3647.

[Geib et al., 2006] Geib, C., Mourão, K., Petrick, R. P. A., Pugeault, N., Steedman, M., Krueger, N., and Wörgötter, F. (2006). Object action complexes as an interface for planning and robot control. In *Workshop: Towards Cognitive Humanoid Robots at IEEE RAS*.

[Getoor and Taskar, 2007] Getoor, L. and Taskar, B. (2007). *Introduction to statistical relational learning*. The MIT press.

[Gibson, 1979] Gibson, J. J. (1979). *The Ecologial Approach to visual perception*. Boston: Houghton Mifflin.

[Gienger et al., 2008] Gienger, M., Toussaint, M., and Goerick, C. (2008). Task maps in humanoid robot manipulation. In *Intelligent Robots and Systems (IROS), 2008 IEEE/RSJ International Conference on*.

[Goodman et al., 2008] Goodman, N., Mansinghka, V. K., Roy, D. M., Bonawitz, K., and Tenenbaum, J. B. (2008). Church: a language for generative models. In *UAI*, pages 220–229.

[Gutmann et al., 2011a] Gutmann, B., Thon, I., and De Raedt, L. (2011a). Learning the parameters of probabilistic logic programs from interpretations. In *ECML*.

[Gutmann et al., 2011b] Gutmann, B., Thon, I., Kimmig, A., Bruynooghe, M., and Raedt, L. D. (2011b). The magic of logical inference in probabilistic programming. *CoRR*, abs/1107.5152.

[Heckerman et al., 1995] Heckerman, D., Geiger, D., and Chickering, D. M. (1995). Learning bayesian networks: The combination of knowledge and statistical data. *Machine learning*, 20(3):197–243.

[Hertzberg and Chatila, 2008] Hertzberg, J. and Chatila, R. (2008). AI reasoning methods for robotics. In *Handbook of Robotics*, pages 207–223. Springer.

[Hirschmuller, 2008] Hirschmuller, H. (2008). Stereo processing by semiglobal matching and mutual information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2):328–341.

[Horton et al., 2012] Horton, T. E., Chakraborty, A., and St. Amant, R. (2012). Affordances for robots: a brief survey. *AVANT. Pismo Awangardy Filozoficzno-Naukowej*, (2):70–84.

[Jain et al., 2009] Jain, D., Mösenlechner, L., and Beetz, M. (2009). Equipping robot control programs with first-order probabilistic reasoning capabilities. In *Robotics and Automation (ICRA), 2009 IEEE International Conference on*, pages 3626–3631.

[Jain and Inamura, 2013] Jain, R. and Inamura, T. (2013). Bayesian learning of tool affordances based on generalization of functional feature to estimate effects of unseen tools. *Artificial Life and Robotics*, 18(1-2):95–103.

[Jetchev and Toussaint, 2010] Jetchev, N. and Toussaint, M. (2010). Trajectory prediction in cluttered voxel environments. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*.

[Joho et al., 2011] Joho, D., Senk, M., and Burgard, W. (2011). Learning search heuristics for finding objects in structured environments. *RAS*, 59(5):319–328.

[Kaelbling et al., 1998] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134.

[Katz et al., 2009] Katz, D., Pyuro, Y., and Brock, O. (2009). Learning to manipulate articulated objects in unstructured environments using a grounded relational representation. In *Robotics: Science and Systems IV*, page 254. MIT Press.

[Kearns et al., 2002] Kearns, M., Mansour, Y., and Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49(2-3):193–208.

[Kjærulff and Madsen, 2005] Kjærulff, U. B. and Madsen, A. L. (2005). Probabilistic networks - an introduction to Bayesian networks and influence diagrams. *Aalborg University*.

[Kocsis and Szepesvári, 2006] Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *ECML*, pages 282–293.

[Kollar and Roy, 2009] Kollar, T. and Roy, N. (2009). Utilizing object-object and object-scene context when planning to find things. In *Robotics and Automation (ICRA), 2009 IEEE International Conference on*, pages 2168–2173. IEEE.

[Koren and Koren, 1985] Koren, Y. and Koren, Y. (1985). *Robotics for engineers*, volume 168. McGraw-Hill New York et al.

[Koski and Noble, 2012] Koski, T. J. and Noble, J. (2012). A review of bayesian networks and structure learning. *Mathematica Applicanda*, 40(1):51–103.

[Kraft et al., 2008] Kraft, D., Pugeault, N., Başeski, E., Popović, M., Kragić, D., Kalkan, S., Wörgötter, F., and Krüger, N. (2008). Birth of the object: detection of objectness and extraction of object shape through object–action complexes. *International Journal of Humanoid Robotics*, 5(02):247–265.

[Kragic et al., 2005] Kragic, D., Björkman, M., Christensen, H. I., and Eklundh, J. O. (2005). Vision for robotic object manipulation in domestic settings. *Robotics and Autonomous Systems*, 52(1):85–100.

[Krüger et al., 2011] Krüger, N., Geib, C., Piater, J., Petrick, R., Steedman, M., Wörgötter, F., Ude, A., Asfour, T., Kraft, D., Omrčen, D., Agostini, A., and Dillmann, R. (2011). Object–Action complexes: Grounded abstractions of sensory–motor processes. *Robotics and Autonomous Systems*, 59(10):740–757.

[Kruger et al., 2009] Kruger, N., Piater, J., Worgotter, F., Geib, C., Petrick, R., Steedman, M., Asfour, T., Kraft, D., Hommel, B., Agostini, A., Kragic, D., Eklundh, J.-O., Kruger, V., Torras, C., and Dillmann, R. (2009). A formal definition of object-action complexes and examples at different levels of the processing hierarchy. *Computer and Information Science*, pages 1–39.

[Krunic et al., 2009] Krunic, V., Salvi, G., Bernardino, A., Montesano, L., and Santos-Victor, J. (2009). Affordance based word-to-meaning association. In *Robotics and Automation (ICRA), 2009 IEEE International Conference on*.

[Kushmerick et al., 1995] Kushmerick, N., Hanks, S., and Weld, D. S. (1995). An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):239–286.

[Lang and Toussaint, 2009] Lang, T. and Toussaint, M. (2009). Approximate inference for planning in stochastic relational worlds. In *ICML*, pages 585–592.

[Lang and Toussaint, 2010a] Lang, T. and Toussaint, M. (2010a). Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research (JAIR)*, 39:1–49.

[Lang and Toussaint, 2010b] Lang, T. and Toussaint, M. (2010b). Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research*, 39:1–49.

[Lloyd, 1987] Lloyd, J. (1987). *Foundations of Logic Programming*. Berlin: Springer-Verlag.

[Lloyd, 1982] Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.

[Lopes et al., 2007] Lopes, M., Melo, F. S., and Montesano, L. (2007). Affordance-based imitation learning in robots. In *Intelligent Robots and Systems (IROS), 2007 IEEE/RSJ International Conference on*, pages 1015–1021.

[Lorken and Hertzberg, 2008] Lorken, C. and Hertzberg, J. (2008). Grounding planning operators by affordances. In *International Conference on Cognitive Systems*.

[Lungarella et al., 2003] Lungarella, M., Metta, G., Pfeifer, R., and Sandini, G. (2003). Developmental robotics: A survey. *Connection Science*, 15:151–190.

[Mcdermott et al., 1998] Mcdermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL - The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

[Mehta et al., 2011] Mehta, N., Tadepalli, P., and Fern, A. (2011). Autonomous learning of action models for planning. In *Advances in Neural Information Processing Systems*, pages 2465–2473.

[Meltzoff and Moore, 1997] Meltzoff, A. N. and Moore, M. K. (1997). Explaining facial imitation: A theoretical model. *Early Development & Parenting*, 6(3-4):179.

[Metta et al., 2006] Metta, G., Fitzpatrick, P., and Natale, L. (2006). Yarp: Yet another robot platform. *International Journal on Advanced Robotics Systems*, 3(1):43–48.

[Metta et al., 2008] Metta, G., Sandini, G., Vernon, D., Natale, L., and Nori, F. (2008). The iCub humanoid robot: an open platform for research in embodied cognition. In *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pages 50–56. ACM.

[Milch et al., 2005] Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D., and Kolobov, A. (2005). BLOG: Probabilistic models with unknown objects. In *IJCAI*, pages 1352–1359.

[Moldovan et al., 2013a] Moldovan, B., Antanas, L., and Hoffmann, M. (2013a). Opening doors: An initial SRL approach. In *Lecture Notes in Computer Science, ILP (International Conference on Inductive Logic Programming), Dubrovnik, Croatia, 17-19 September 2012*, pages 178–192. Springer.

[Moldovan and De Raedt, 2014a] Moldovan, B. and De Raedt, L. (2014a). Learning relational affordance models for two-arm robots. In *Intelligent Robots and Systems (IROS), 2014 IEEE/RSJ International Conference on.*

[Moldovan and De Raedt, 2014b] Moldovan, B. and De Raedt, L. (2014b). Occluded object search by relational affordances. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on.*

[Moldovan et al., 2012a] Moldovan, B., Moreno, P., van Otterlo, M., Santos-Victor, J., and De Raedt, L. (2012a). Learning relational affordance models for robots in multi-object manipulation tasks. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on.*

[Moldovan et al., 2013b] Moldovan, B., Thon, I., Davis, J., and De Raedt, L. (2013b). MCMC estimation of conditional probabilities in probabilistic programming languages. In *Lecture Notes in Computer Science, ECSQARU (European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty), Utrecht, The Netherlands, 7-10 July 2013*, pages 436 –448. Springer.

[Moldovan et al., 2012b] Moldovan, B., van Otterlo, M., Moreno, P., Santos-Victor, J., and De Raedt, L. (2012b). Statistical relational learning of object affordances for robotic manipulation. *Latest advances in inductive logic programming*, page 6.

[Montesano et al., 2008] Montesano, L., Lopes, M., Bernardino, A., and Santos-Victor, J. (2008). Learning object affordances: From sensory-motor coordination to imitation. *IEEE Transactions on Robotics*, 24:15–26.

[Mörwald et al., 2010] Mörwald, T., Prankl, J., Richtsfeld, A., Zillich, M., and Vincze, M. (2010). BLORT - The Blocks World Robotic Vision Toolbox. In *Best Practice in 3D Perception and Modeling for Mobile Manipulation*.

[Mourão et al., 2010] Mourão, K., Petrick, R. P. A., and Steedman, M. (2010). Learning action effects in partially observable domains. In *ECAI*, pages 973–974.

[Murata et al., 1997] Murata, A., Fadiga, L., Fogassi, L., Gallese, V., Raos, V., and Rizzolatti, G. (1997). Object representation in the ventral premotor cortex (area F5) of the monkey. *Journal of neurophysiology*, 78(4):2226–2230.

[Murphy et al., 2001] Murphy, K. et al. (2001). The Bayes net toolbox for Matlab. *Computing science and statistics*, 33(2):1024–1034.

[Murphy, 2002] Murphy, K. P. (2002). *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley.

[Nitti et al., 2014] Nitti, D., De Laet, T., and De Raedt, L. (2014). Relational object tracking and learning. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*.

[Nitti et al., 2013] Nitti, D., Laet, T. D., and Raedt, L. D. (2013). A particle filter for hybrid relational domains. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*.

[Pasula et al., 2004] Pasula, H. M., Zettlemoyer, L. S., and Kaebling, L. P. (2004). Learning probabilistic relational planning rules. In *ICAPS*, pages 73–82.

[Pattacini et al., 2010] Pattacini, U., Nori, F., Natale, L., Metta, G., and Sandini, G. (2010). An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*.

[Pearl, 1988] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

[Pfeffer, 2001] Pfeffer, A. (2001). IBAL : A probabilistic rational programming language. In *IJCAI*, pages 733–740.

[Pieropan et al., 2014] Pieropan, A., Ek, C. H., and Kjellström, H. (2014). Recognizing object affordances in terms of spatio-temporal object-object relationships. *IROS, In submission*.

[Poole, 1997] Poole, D. (1997). The independent choice logic for modelling multiple agents under uncertainty. *Artif. Intell*, 94(1-2):7–56.

[Rizzolatti and Craighero, 2004] Rizzolatti, G. and Craighero, L. (2004). The mirror-neuron system. *Annu. Rev. Neurosci.*, 27:169–192.

[Rodrigues et al., 2011] Rodrigues, C., Gérard, P., Rouveirol, C., and Soldano, H. (2011). Relational action model learning and planning integration. *7e Plateforme AFIA*, page 361.

[Rome et al., 2006] Rome, E., Doherty, P., Dorffner, G., and Hertzberg, J. (2006). Towards affordance-based robot control. In *Towards Affordance-Based Robot Control*, number 06231 in Dagstuhl Seminar Proceedings.

[Russell and Norvig, 2010] Russell, S. J. and Norvig, P. (2010). *Artificial Intelligence - A Modern Approach*. Pearson Education.

[Sablon and Bruynooghe, 1994] Sablon, G. and Bruynooghe, M. (1994). Using the event calculus to integrate planning and learning in an intelligent autonomous agent. In Bäckström, C. and Sandewall, E., editors, *Current Trends in AI Planning*, pages 254–265. IOS Press.

[Saegusa et al., 2009] Saegusa, R., Metta, G., Sandini, G., and Sakka, S. (2009). Active motor babbling for sensorimotor learning. In *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*, pages 794–799. IEEE.

[Sahin and Erdogan, 2009] Sahin, E. and Erdogan, S. T. (2009). Towards linking affordances with mirror/canonical neurons. In *Computer and Information Sciences, 2009. ISCIS 2009. 24th International Symposium on*. IEEE.

[Samadi et al., 2012] Samadi, M., Kollar, T., and Veloso, M. (2012). Using the web to interactively learn to find objects. In *AAAI*. AAAI Press.

[Sanghai et al., 2005] Sanghai, S., Domingos, P., and Weld, D. (2005). Relational dynamic bayesian networks. *Journal of Artificial Intelligence Research*, pages 759–797.

[Sato, 1995] Sato, T. (1995). A statistical learning method for logic programs with distribution semantics. In *ICLP*, pages 715–729.

[Sato and Kameya, 1997] Sato, T. and Kameya, Y. (1997). PRISM: A language for symbolic-statistical modeling. In *IJCAI*, pages 1330–1339.

[Schuster et al., 2012] Schuster, M., Jain, D., Tenorth, M., and Beetz, M. (2012). Learning organizational principles in human environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*.

[Shahaf et al., 2006] Shahaf, D., Chang, A., and Amir, E. (2006). Learning partially observable action models: Efficient algorithms. In *Proceedings of the national conference on Artificial Intelligence*, volume 21, page 920. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

[Shubina and Tsotsos, 2010] Shubina, K. and Tsotsos, J. (2010). Visual search for an object in a 3D environment using a mobile robot. *CVIU*, 114(5):535–547.

[Sinapov and Stoytchev, 2007] Sinapov, J. and Stoytchev, A. (2007). Learning and generalization of behavior-grounded tool affordances. In *ICDL*, pages 19–24.

[Sinapov and Stoytchev, 2011] Sinapov, J. and Stoytchev, A. (2011). Object category recognition by a humanoid robot using behavior-grounded relational learning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 184–190. IEEE.

[Sjö et al., 2009] Sjö, K., López, D., Paul, C., Jensfelt, P., and Kragic, D. (2009). Object search and localization for an indoor mobile robot. *CIT*, 17:67–80.

[Steedman, 2002] Steedman, M. (2002). Formalizing affordance. In *Annual Meeting of the Cognitive Science Society*, pages 834–839.

[Stoytchev, 2005] Stoytchev, A. (2005). Behavior-grounded representation of tool affordances. In *Robotics and Automation (ICRA), 2005 IEEE International Conference on*, pages 3060–3065.

[Stulp and Beetz, 2008] Stulp, F. and Beetz, M. (2008). Combining declarative, procedural, and predictive knowledge to generate, execute, optimize robot plans. *Robotics and Autonomous Systems*, 56:967–979.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

[Szedmak et al., 2014] Szedmak, S., Ugur, E., and Piater, J. (2014). Knowledge propagation and relation learning for predicting action effects. In *Intelligent Robots and Systems (IROS), 2014 IEEE/RSJ International Conference on*, pages 623–629. IEEE.

[Tenorth and Beetz, 2009] Tenorth, M. and Beetz, M. (2009). KNOWROB - knowledge processing for autonomous personal robots. In *Intelligent Robots and Systems (IROS), 2009 IEEE/RSJ International Conference on*, pages 4261–4266. IEEE.

[Thrun et al., 2005] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press.

[Thrun et al., 2006] Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., et al. (2006). Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692.

[Toussaint et al., 2010] Toussaint, M., Plath, N., Lang, T., and Jetchev, N. (2010). Integrated motor control, planning, grasping and high-level reasoning in a blocks world using probabilistic inference. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*.

[Turvey, 1992] Turvey, M. T. (1992). Affordances and prospective control: An outline of the ontology. *Ecological psychology*, 4(3):173–187.

[Ugur et al., 2007] Ugur, E., Dogar, M. R., Cakmak, M., and Sahin, E. (2007). The learning and use of traversability affordance using range images on a mobile robot. In *Robotics and Automation (ICRA), 2007 IEEE International Conference on*, pages 1721–1726.

[Ugur et al., 2009] Ugur, E., Sahin, E., and Oztop, E. (2009). Affordance learning from range data for multi-step planning. In *International Conference on Epigenetic Robotics (EpiRob)*.

[Ugur et al., 2014] Ugur, E., Szedmak, S., and Piater, J. (2014). Bootstrapping paired-object affordance learning with learned single-affordance features. In *4th International Conference on Development and Learning and on Epigenetic Robotics*.

[Vahrenkamp et al., 2009] Vahrenkamp, N., Berenson, D., Asfour, T., Kuffner, J., and Dillmann, R. (2009). Humanoid motion planning for dual-arm manipulation and re-grasping tasks. In *Intelligent Robots and Systems (IROS), 2009 IEEE/RSJ International Conference on*, pages 2464–2470.

[van Otterlo, 2009] van Otterlo, M. (2009). *The Logic of Adaptive Behavior*. IOS Press, Amsterdam, The Netherlands.

[Vennekens et al., 2009] Vennekens, J., Denecker, M., and Bruynooghe, M. (2009). CP-logic: A language of causal probabilistic events and its relation to logic programming. *TPLP*, 9(3):245–308.

[Wiering and van Otterlo, 2012] Wiering, M. A. and van Otterlo, M. (2012). *Reinforcement Learning: State-of-the-Art*. Springer.

[Wong et al., 2013] Wong, L., Kaelbling, L., and Lozano-Peréz, T. (2013). Manipulation-based active search for occluded objects. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*.

[Wörgötter et al., 2009] Wörgötter, F., Agostini, A., Krüger, N., Shylo, N., and Porr, B. (2009). Cognitive agents – a procedural perspective relying on the predictability of object-action complexes (OACs). *Robotics and Autonomous Systems*, 57:420–432.

[Yang et al., 2007] Yang, Q., Wu, K., and Jiang, Y. (2007). Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171(2):107–143.

[Ye and Tsotsos, 1996] Ye, Y. and Tsotsos, J. (1996). Sensor planning in 3D object search: Its formulation and complexity. *Annals of Mathematics and AI*.

[Younes and Littman, 2004] Younes, H. L. and Littman, M. L. (2004). Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects.

[Zettlemoyer et al., 2005] Zettlemoyer, L. S., Pasula, H., and Kaelbling, L. P. (2005). Learning planning rules in noisy stochastic worlds. In *AAAI*, pages 911–918.

[Zöllner et al., 2004] Zöllner, R., Asfour, T., and Dillmann, R. (2004). Programming by demonstration: dual-arm manipulation tasks for humanoid robots. In *Intelligent Robots and Systems (IROS), 2004 IEEE/RSJ International Conference on*, pages 479–484.

Curriculum Vitae

Bogdan Moldovan studied Computer Engineering at the University of Waterloo, Ontario, Canada. In June 2005 he obtained a Bachelor of Applied Science in Honours Computer Engineering, Software Engineering Option.

In 2005 he joined KTH (Kungliga Tekniska Högskolan) in Stockholm, Sweden, for a Master of Science with a major in Information Technology, specialization in Software Engineering of Distributed Systems. He received his MSc diploma in 2007 with the thesis titled "Distributed Execution Framework for Test and Verification Systems".

In March 2010 he joined the Declarative Languages and Artificial Intelligence (DTAI) group at KU Leuven for doctoral studies under the supervision of Prof. Luc De Raedt. In January 2011 he received a 4 year scholarship from IWT (agentschap voor Innovatie door Wetenschap en Technologie) to work on his PhD on probabilistic programming for robotics.

List of publications

Journal articles

- B. Moldovan, P. Moreno, D. Nitti, J. Santos-Victor, L. De Raedt. *Using Relational Affordances for Multiple-Action Two-Arm Manipulation Tasks*, to be submitted to the Robotics and Autonomous Systems journal

Conference papers

- B. Moldovan, P. Moreno, M. van Otterlo, J. Santos-Victor, L. De Raedt. *Statistical Relational Learning of Object Affordances for Robotic Manipulation*, in Latest Advances in Inductive Logic Programming, 2012
- B. Moldovan, P. Moreno, M. van Otterlo, J. Santos-Victor, L. De Raedt. *Learning Relational Affordance Models for Robots in Multi-Object Manipulation Tasks*, in Proceedings of the 29th IEEE International Conference on Robotics and Automation (ICRA), St. Paul, MN, USA, 2012
- B. Moldovan, L. Antanas, M. Hoffmann. *Opening doors: An initial SRL approach*, in Lecture Notes in Computer Science, International Conference on Inductive Logic Programming (ILP), Dubrovnik, Croatia, 2012
- B. Moldovan, P. Moreno, M. van Otterlo. *On the Use of Probabilistic Relational Affordance Models for Sequential Manipulation Tasks in Robotics*, in Proceedings of the 30th IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany, 2013
- B. Moldovan, I. Thon, J. Davis, L. De Raedt. *MCMC estimation of conditional probabilities in probabilistic programming languages*, in Lecture Notes in Computer Science, European Conference on Symbolic

and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU), Utrecht, The Netherlands, 2013

- B. Moldovan, L. De Raedt. *Occluded Object Search by Relational Affordances*, in Proceedings of the 31st IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 2014
- B. Moldovan, L. De Raedt. *Learning Relational Affordance Models for Two-Arm Robots*, in Proceedings of the 27th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Chicago, IL, USA, 2014

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
DECLARATIVE LANGUAGES AND ARTIFICIAL INTELLIGENCE
Celestijnenlaan 200A
B-3001 Heverlee, Belgium
tel. +32 16 32 7700
bogdan.moldovan@cs.kuleuven.be
dtai.cs.kuleuven.be